AFIT/GOR/ENS/99M-06

PILOT INVENTORY COMPLEX ADAPTIVE SYSTEM (PICAS): AN ARTIFICIAL LIFE APPROACH TO MANAGING PILOT RETENTION

THESIS

Martin P. Gaupp, Captain, USAF

AFIT/GOR/ENS/99M-06

AFIT/GOR/ENS/99M-06

PILOT INVENTORY COMPLEX ADAPTIVE SYSTEM (PICAS): AN ARTIFICIAL LIFE APPROACH TO MANAGING PILOT RETENTION

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Martin P. Gaupp, B.S., M.B.A.

Captain, USAF

March 1999

AFIT/GOR/ENS/99M-06

PILOT INVENTORY COMPLEX ADAPTIVE SYSTEM (PICAS): AN ARTIFICIAL LIFE APPROACH TO MANAGING PILOT RETENTION

THESIS

Martin P. Gaupp, Captain, USAF

Approved:

Raymond R Hill Jr — 10 Mar 99

Raymond Hill (chairman) — date

[illegible] — 10 Mar 99

Glenn Bailey (reader) — date

ACKNOWLEDGEMENTS

This thesis effort would never have even been started without the help of my faculty advisor, Major Ray Hill. His guidance and patience with me throughout the entire process made this project truly enjoyable and very enlightening. I would also like to thank my reader, Lt Col Glenn Bailey, for being there to bounce ideas off of and encouraging me in completing this effort. My sponsor, Major Craig Willits, was always there to help too, be it either with finances or knowledge. Thanks to his quick responses to my questions I never had to experience the data-collection delays that seem to plaque AFIT theses.

Above all, though, I thank my wife Britta. Her patience over the last 18 months here at AFIT made this grueling effort that much easier. Without her unwavering love and support none of this would have been possible or even purposeful. I fear that I take for granted too often all that she puts up with while I am in school and therefore am for ever indebted to her.

Lastly I thank God for the opportunity He provided me with here at AFIT. His blessing upon my life and the work He has allowed me to do here are a true testament to what faith can do if you only believe.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The retention of skilled pilots continues to be a problem that plagues the United States Air Force. After spending millions of dollars on training and education, it is disheartening to see the mass exodus of experienced aviators from the Air Force that has been occurring in the past decade. Many blame the economy, others the Air Force itself, but few are able to accurately predict how or why they are all leaving. The current personnel models do not adequately determine retention rates. Complex adaptive systems theory, however, might provide some insight. By modeling the system at the pilot's level, allowing each pilot to be represented as an autonomous, independent agent continually adapting to its environment and the other agents in it, an alternate model can be built; one that accounts for the interactions among the pilots, not just their interactions with their environment. PICAS (Pilot Inventory Complex Adaptive System) is just such a model. Constructed in the Java language, the PICAS model exploits the notions of complex adaptive systems theory and employs dynamic user controls to discern retention rates. Pilots 'evolve', for lack of a better word, to a greater fitness within their environment, and in the process the model user can better determine what kind of environment needs to be created and maintained in order to ensure that trained and experienced pilots are in fact retained for their services. This thesis will discuss the theory underlying the PICAS model, trace the development of the PICAS model, review the descriptive results the model produces, and finally investigate the uses of the model as a tool for exploratory modeling.

PILOT INVENTORY COMPLEX ADAPTIVE SYSTEM (PICAS): AN ARTIFICIAL LIFE APPROACH TO MANAGING PILOT RETENTION

CHAPTER 1

1.1. Overview of the Thesis Effort

1.1.1. Definition of the Topic and Problem Statement

During the draw-down of the early 1990's, personnel retention was not a real issue. The military was interested in reducing its total force levels and losing some people due to separations was actually advantageous. However, the draw-down is now over. The military is at a stable force structure, but personnel are still separating at an alarming rate. Most notably, pilot retention has become a major problem for the Air Force. After investing millions on their training, the Air Force is finding it increasingly harder to maintain the necessary pilot personnel levels. Too many pilots are taking their Air Force gained knowledge and experience and parlaying it into a career in the commercial sector. The Air Force does not totally understand why so many pilots are leaving nor how to reverse the trend. Current personnel models adequately predict short-term inventory levels but really cannot account for the mass exodus occurring and ultimately are not intended to provide indications on how to solve the problem.

This is where the notion of complex adaptive systems (CAS) theory comes in. By looking at the pilot retention issue from a complex adaptive systems standpoint, we model pilots as individual, autonomous, adaptive agents. Essentially, we model pilot retention from the standpoint of the pilots, incorporating the interactions among the pilots and their environment to examine the overall system's behavioral response. The resulting model might then provide us with insights into the issues and motivations underlying pilot retention. This information can then be used by policy makers to tailor their decisions in order to minimize the separation of trained pilots early in their Air Force careers.

1.1.2. Scope of the Research

This research centers around the creation of a complex adaptive systems simulation model using the Java programming language. Developed as an applet executable on any Java-enabled web browser, the model consists of a group of simulated pilots that are treated as autonomous, independently thinking agents. Each pilot is preprogrammed with a simple set of rules and then set loose to interact with other pilots in the system (i.e. squadron, wing, or AF-wide) and react to the environment. The pilots' interactions with each other and their environment is what leads to some interesting emergent behavior regarding their tendencies to stay in the Air Force (aka retention) or their tendencies to leave the Air Force (aka separation).

The prototype model, called PICAS (Pilot Inventory Complex Adaptive System) is then used in an exploratory analysis role. A designed experiment is run and response surfaces are generated, suggesting a methodology for insight producing analyses using CAS-based simulations.

1.2. Format of this Thesis

This thesis is structured in a way to make publication in an academic journal as simple as possible. Chapter 2 contains all the information regarding the model's conceptual development and its implementation in the Java language. Chapter 2 also presents model results and suggests potential uses for the model in real-world applications. Chapter 3 provides a conclusion and presents areas for further research.

The appendices provide additional background information. Appendix A presents an in-depth review of chaos theory, complex adaptive systems theory, and artificial life. Included in each section are the history and development of the theory, the basic tenets of the theory, and the various real-world applications of the theory. Appendix B provides the source code of the model developed for this thesis. Appendix C contains an annotated bibliography of the various sources consulted in researching this thesis.

CHAPTER 2

2.1. Introduction

Life is complex. But, since the dawn of time, humans have tried to eliminate the complexity in our lives. We concoct assumptions to relax the rigid requirements of a complicated system. In model building, we eliminate the complex to simplify the problem and promote understanding. In reality, however, we sacrifice reality for simplicity. We have models that work, but often they over abstract reality and end up answering the wrong questions. Such models often reconfirm common knowledge.

According to Casti (1994:4), models are nothing but an applied set of rules. The problem, he points out, is for too long we have been building our models by taking apart existing systems and then dictating from the top down how to put them back together based on the knowledge of their individual parts. We then expect our recreated whole to be an adequate representation of the real whole we are studying. The problem is, as Aristotle points out, "The whole is more than the sum of the parts" (quoted in Casti, 1994:171). According to Casti, "if you want to study the behavior of a system composed of many parts, breaking it apart into its component pieces and studying the pieces separately won't always help you in understanding the whole" (Casti, 1994:172).

Current personnel models are either regression-based predictive models or entity-based descriptive models. While sufficient for forecasting and predictive purposes, these models do not explicitly capture personnel behaviors, especially with regards to the interactions between personnel. Capturing personnel behavior is simply not within the scope of current models. To gain analytical insight into personnel behavior trends and exploit the interactive nature of the actual system, a new generation of models may be required.

This is where complex adaptive systems (CAS) theory may be able to help. Instead of modeling from the top-down—the current paradigm—CAS implements bottom-up modeling. Agents are created in the model and their interactions with each other and their environment is what drives the system's behavior. Observing the trends in the systems behavior can then help describe past behavior and lend some insight into predicting future behavior.

The goal of the PICAS (Pilot Inventory Complex Adaptive System) model is to create artificial life in the computer to such a degree that the actions of the pilots in the computer resemble the expected separation actions of real-life Air Force pilots. The computer's pilots are subjected to environmental conditions in the hope that their reactions will provide some indication of the behaviors exhibited by real pilots in similar situations. Controlling conditions and observing behaviors is easier in such a model than doing so in real-life.

This research is a 'proof-of-concept' effort examining complex adaptive systems applicability to Air Force modeling. Section 2 overviews CAS theory followed by an overview of the conceptual model in section 3. Section 4 covers the actual implementation of the model in the Java programming language and section 5 discusses verification and validation issues. Analysis methods are described in section 6 and results are presented in section 7. We then close with a summary and some concluding remarks.

2.2. Complex Adaptive Systems

To break away from the paradigm of top-down modeling, we enter the science of complexity and its applications in complex adaptive systems theory. Instead of the top-down approach so entrenched in contemporary simulation and modeling, complex adaptive systems theory is based on the notion of model building from the bottom up.

> After three hundred years of dissecting everything into molecules and atoms and nuclei and quarks, they [scientists] finally seemed to be turning that process inside out. Instead of looking for the simplest pieces possible, they were starting to look at how those pieces go together into complex wholes. (Waldrop, 1992:16)

As the idea of emergence and learning became popular, scientists began to realize that dissecting a system into its parts was not an adequate way to describe and reproduce its behavior. There was something inherent in the entire entity that its parts alone could not describe. There was something hidden in the whole that the parts could not reveal by themselves. Scientists were discovering that complexity was really more a property of the interactions between the parts of a system than an intrinsic aspect of the parts themselves taken in isolation.

Scientists also uncovered the notion that complex behavior seemed to emerge from these complicated systems. Emergence, it turns out, is something that we humans have been exposed to for centuries, but have always taken for granted. Holland points out that:

> We are everywhere confronted with emergence in complex adaptive systems—ant colonies, networks of neurons, the immune system, the Internet, and the global economy, to name a few—where the behavior of the whole is much more complex than the behavior of the parts. (Holland, 1995:2)

Holland goes on to discuss the nature of these emergent systems. He states that:

> A small number of rules or laws can generate systems of surprising complexity. Moreover, this complexity is not just the complexity of random patterns. Recognizable features exist… In addition, the systems are animated—dynamic; they change over time. Though the laws are invariant, the things they govern change… The rules or laws generate the complexity, and the ever-changing flux of patterns that follows leads to perpetual novelty and emergence. (Holland, 1995:3-4)

Emergent behavior seems to appear out of nowhere. Added to this, the notion of such behavior existing is not even a consideration among the individual agents that make up the system. The notion of the emergent behavior is strictly tied to the actions of the entire system. Once again, the whole is greater than the sum of the parts.

With this in mind, the science of complexity began growing and gaining notoriety. Members from all over the scientific community, from astronomy to zoology, were coming together and contributing to the formation of this new science. The Santa Fe Institute was founded as a private organization chartered with research into the phenomena of complex adaptive systems. There, scientists from throughout the world gathered and began to form the new science of complexity.

From the outset, a definition of complexity had to be developed before any real experimentation or model building could occur. The members of the Santa Fe Institute went to work to develop the characteristics they felt were necessary to describe a complex adaptive system. First, it was agreed that complex processes generate counterintuitive, seemingly acausal behavior which is full of surprises. This alone was a disturbing fact and one that would require many paradigm shifts in the scientific community. It was also discovered that complex systems involve lots of interactions among the entities in the system and feedback plays a big role in the system's overall behavior. Complex systems also exhibit a marked diffusion of real authority. Contrary to current systems theory which believed that every system seemed to have a nominal supreme decision-maker, complex adaptive systems possess a decentralized power structure spreading authority and decision making ability over a number of units (if not all the units in the system). System behavior is the combination of the individual behaviors of the agents acting together. Thus, complex adaptive systems might simulate life, leading to the notion of exploiting complex adaptive systems theory in the investigation of artificial life. Lastly, complex adaptive systems are inherently irreducible. Neglecting any part of the process or severing any of the connections linking its parts would destroy essential aspects of the system's behavior or structure and render the system inert. Again, complexity was a property of the entire system, not the individual parts of the system.

New models arose exploiting the tenets of complex adaptive systems theory. However, a chief argument arose about the predictive power of such models. Complex adaptive systems rarely (if ever) followed the same course of action even when all the inputs were maintained between simulations. This, however, was a very realistic property that complex adaptive systems engineers were proud of. Life is not predictable; the same initial circumstances can lead to greatly diverse outcomes. With this in mind, the creators of models employing complex adaptive systems theory held on to the notion that models are not just important for their predictive abilities. Their ability to explain natural phenomena is just as important. Users of complex adaptive systems were convinced their models might help describe nature, something no other models had been able to accomplish. Although they may not be able to predict the future state of a system, they might be able to describe, and re-create, its past. This alone would be a great accomplishment, for to date not many models had been successful at explaining anything but the most simple of systems.

Complex adaptive system theory helped develop the notion of artificial life. Levy says "Artificial life, or a-life, is devoted to the creation and study of lifelike organisms and systems built by humans" (Levy, 1992:5). Langton adds that:

> The ultimate goal of the study of artificial life would be to create "life" in some other medium, ideally a virtual medium where the essence of life has been abstracted from the details of its implementation in any particular model. We would like to build models that are so life-like that they cease to become models of life and become examples of life themselves. (quoted in Levy, 1992:85)

2.3. Conceptual Model Development

2.3.1. Pilots as Complex Adaptive Agents

The pilot inventory problem in the Air Force is extremely complicated. There are numerous constraints and many different variables that play into the problem. However, underneath all the details lies the essence of a complex adaptive system. The pilots themselves are actually agents adapting to better survive in a complex environment. Capturing the autonomy with which these pilot-agents act within a model requires that we exploit the tenets of complex adaptive systems theory. Pilots constantly communicate with each other and influence each other's actions. They take in information from their environment, process it, and then react to it. They also take in information regarding the behaviors of other pilots in their vicinity (i.e., squadron-mates) and base their behaviors on this interactive information. This alone defines the nature of a complex adaptive system and it is based on this notion that the idea of using a complex adaptive systems theory to model the pilot inventory issue arose.

The whole purpose of model building is to explain some complicated process in a simplified manner without sacrificing too much information. Holland points out that a well constructed model exhibits the complexity and emergent behavior of the system it models but without all the (often unnecessary) detail involved in the real system (Holland, 1998:12). A model should simplify the system, but not to the point of loosing any of the significant information inherent in the system being modeled.

The PICAS model was developed to exploit the complex adaptive systems aspects of the pilot inventory issue. By setting up a few simple rules by which the agents (pilots) in the model interact, some very complex and interesting behavior emerges. The interactions between the agents cause the emergence of complex behavior, behavior that may very well point out ways to avoid or exploit future scenarios in the personnel management arena.

Artificial life is also important since the whole goal of the model is to simulate the life-like behavior of pilots in the Air Force. The agents in the model are constructed in such a way that their emergent behavior appears life-like—it mimics their counterparts outside the computer. By altering the computer agent's environment and agent-to-agent behavior, the users of the model may gain insights into how a real pilot might react under the same circumstances. In this way, the computer's pilot-agents may be viewed as simplified examples of real Air Force pilots, although numerous simplifying assumptions underlie the current prototype PICAS model.

2.3.2. The Pilots' Environment

The environment within which the pilots live is represented by the playing field on the screen. In the real world, the pilots' environment is constantly changing and pilots are always trying to adjust their behaviors in order to maximize their happiness in their current environment. A similar idea is used by PICAS. The environment within which the pilot-agents live is also in a constant state of flux and the agents are continuously adapting themselves to their new virtual environment.

The goal was to develop an environment capturing the essential information present in the pilots' real environment. However, due to the computer's ability to process lots of information in short amounts of time, the computer's environment can be changed and altered at the whim of the user. This allows the user to investigate the effects of certain policy decisions without having to wait and see what their effects are in the real system. By modeling the system this way, the user is in total control of the environment (something impossible in the real world) and is better able to determine the possible implications of certain policies and procedures that could be implemented on the real pilot force. By being under the user's control, the dynamic nature of the model allows the user to test personnel policy initiatives without risking harm to the real system. Results are obtained in a fraction of the time and at a significantly reduced price than if tested on the real system.

2.3.3. Collection of Summary Statistics

Throughout the model's execution, the pilots are animated on the playing field, dancing about their environment and making decisions regarding their future career aspirations in the Air Force. Animation, however interesting, does not provide hard quantifiable data for analysis. Animation, though, does provide a means to view the emergent CAS-like behavior exhibited by the model. It allows the user to gain insight into personnel trends and conduct exploratory analysis. To provide quantitative data summary statistics are collected and displayed for the user along the bottom of the screen.

By displaying the percentage of separations at particular times in a pilot's career, the model provides a graphical depiction of the pilots' decision making process and gives the user an ability to view the results of the model over the course of its entire execution run. This information can then be used to guide policy makers and help them determine the best course of action to pursue for the pilot inventory as a whole.

The collection of summary statistics also provides a means by which to compare different executions of the model. As with any simulation, multiple runs produce distributional information about the results. This allows the user to extract more useful and complete information from the model and use it to further refine real-world policy initiatives.

2.3.4. User Interaction

Building the model with extensive user interaction gives the entire system a more appealing feel. The ability to dynamically alter the characteristics of the agents in the model and the environment itself allows the user to conduct 'what-if' type analysis over a great range of parameters while still executing the same run of the model. The user can test both the timing and intensity of his policy decisions and immediately see their effect on the pilots' retention and separation rates.

The parameters over which the user has control can be grouped into two categories, those that affect the behavior of the pilot-agents themselves and those that affect the environment within which the pilot-agents live. The behavior of the actual pilot-agents is controlled by means of shifting a set of two utility curves, one representing the amount of satisfaction that the pilot-agent receives from money, the other representing the amount of gratification the pilot-agent receives from time-off. Added to this, a scroll bar allows the user to determine the relative weight of each utility curve with respect to the other curve, thus allowing for either a money-centered or time-centered attitude (or even if set at 50-50). The environmental settings are controlled by a set of scroll bars. These environmental settings include:

- the amount of pilot jobs available in the commercial airline market;
- the perceived pay gap between military and commercial pilots;
- the current flying operations tempo in the military.

Shifting the scroll bars attached to these settings alters the environment within which the pilot-agents live, thus causing them to alter their behavior and adapt to their new environment. Coupled with the pilot-agents individual attitudes (as measured by their utility curves), the dynamic alteration of the environmental settings allows for the emergence of complex behavior that hopefully mimics trends observed in reality.

User interaction also allows for easier justification of assumptions used within the model. Since few parameters are hard-wired into the model itself, the user is free to adjust the model's settings to fit his assumptions of reality. This facilitates sensitivity analysis. All together, this gives the user more ownership in the entire model and allows him to feel more like a part of the process rather than just a passive observer.

Since PICAS is a proof-of-concept of CAS simulation for personnel analysis, some key assumptions must be clarified. We model pilot preferences using utility curves, a common technique in decision analysis (see Clemen, 1996 and Kirkwood, 1997). Our environmental factor choices are based on what seemed reasonable and what made model use as intuitive as possible. The scales and ranges employed on all slider bars are notional, meant merely to exercise the model. While subsequent analysis, Section 2.6 and beyond, might imply the realism of our parameters, that implication is for analysis methodology development only and should not be construed beyond that use.

2.4. Simulation Model Development in Java

The Java language was used to create the PICAS model in order to maximize the dissemination of the model due to Java's inherent cross platform capabilities. Added to this, the relative ease with which Java handles graphics allowed us to create a simple yet information rich display controlled by an intuitive graphical-user interface (GUI).

The development of an actual working model as a Java applet allows the user to examine in real-time a sample of the type of artificial life-like behavior espoused by complex adaptive systems theory. The use of Java also allows the user to examine the source code with ease and learn about the algorithms used to create the life-like behavior displayed by the model.

2.4.1. Agent Class

The agent class represents the objects being manipulated in the model itself. Each pilot in the model is a separate instantiation of the agent class and has characteristics that are particular to that pilot. The agent class contains information regarding each pilot's location in the environment (by means of a coordinate plane) as well as the parameters that govern the pilot's interactions with other pilots and the environment itself.

The agent class uses a fitness function to help determine the propensity of each pilot's intentions regarding the retention/separation issue. The fitness function is used as a means to determine which portion of the playing field attracts a given pilot-agent. The fitness function combines the environmental parameters with the pilots' attitudes to define an overall satisfaction rating for each agent. The values of the environmental parameters are used as inputs in the utility curves that define the pilots' attitudes. The hiring and paygap parameters are aggregated (via a simple average) and this value is used as the X-axis input to the pilot-agent's money utility curve. The opstempo parameter is used as the X-axis input to the pilot-agent's time utility curve. The Y-axis results from both utility curves are then combined using the ratio specified by the time/money weight scroll bar and a new fitness function is produced. This new fitness function is compared to the pilot-agent's current fitness function and used to adjust the current fitness function either up or down. In this way each pilot-agent gradually adapts to the new environment based on his particular attitude.

The agent class also defines exactly how the pilots interact with each other and their environment. The algorithm used here is based on Reynolds' boids simulation (see Reynolds, 1998) and uses Java code from Dolan's MiniFloys Java applet (see Dolan, 1998). Each pilot-agent in the simulation keeps track of a certain number of neighbors in his vicinity. Based on the agent's fitness and his proximity to other neighbors, he will alter his behavior (i.e., change his movement on the screen) to remain close to his neighbors, while at the same time following his propensity towards retention/separation. A degree of randomness is introduced into the model by 'kicking' the system a certain percentage of the time. This 'kick' results in the random shuffling of neighbors, thus allowing agents to experience effects from other agents that might not otherwise be their closest neighbor. This helps to add reality to the model by simulating the effects of random outside occurrences on a pilot's decision making process (like Permanent Changes of Station [PCSs] or Temporary Duty Assignments [TDYs]).

In the model, each iteration of the algorithm first determines whether or not the current neighbors are in fact the agents closest to a given agent. If not, a swap is done and the closer neighbors are assigned to the agent. Following this, the agent's fitness function is updated based on the current environmental parameters and the current pilot-specific parameters. Then each pilot is processed to determine in which direction and at what speed his next movement should be executed. Reynolds' algorithms (Reynolds, 1998) are used here to ensure that the agents don't get too close to each other (the separation principle), that they try to match the speed and direction of nearby agents (the alignment principle), and that they head for the perceived center of mass of the other agents in the immediate vicinity (the cohesion principle).

Built into the playing field upon which the agents interact are three basins of attraction. The idea of a basin of attraction, first derived by those studying chaos theory, is used as a means to capture the propensity of each agent towards retention or separation. The center of the playing field (the third basin of attraction) is a holding ground for undecided agents. The upper right-hand corner represents retention, and pilots with a propensity to stay in the service migrate to this area when their fitness functions reach a certain level. Conversely, the lower left-hand corner of the playing field represents separation. Pilots whose fitness functions dip below a certain level move to this area of the playing field and eventually leave the system when they separate from the service.

All pilot-agents contain information regarding how many years-of-service (YOS) they have. This parameter is incremented each time the algorithm goes through 100 cycles. Regardless of when the pilot-agents separate from the system, their current years-of-service is recorded for later use in the display of summary statistics. Upon reaching 20 YOS, pilots are removed from the system since they either retire or get promoted to Colonel. This removal bounds the current system. Our rationale is that if a 20 year pilot is not likely to be promoted to Colonel, then he will retire from the Air Force; whereas a 20 year pilot promoted to Colonel will become a Colonel's group asset and be handled by a separate division of the personnel system (which is outside the scope of this model).

2.4.2. PilotRetention Class

This class represents the main applet class that controls the applet's execution. The function of the PilotRetention class is to declare all the applet's variables, initialize the simulation, start the simulation off, control the simulation as it runs, and then stop the simulation in an orderly fashion. These actions are all performed by various methods within the PilotRetention class.

In the initialization method, the PilotRetention class sets up the entire graphical user interface. This includes drawing the playing field, creating the environmental scroll bar controls and the pilot specific utility curves, and setting up the dynamic graph that displays the summary statistics for the user. Lastly, the initialization method instantiates the initial set of pilot-agents and gives the system its starting point.

The start method of this class kicks off the simulation clock. The clock is implemented as the main thread of execution in the program, and uses Java's multithreading capabilities to animate the agents on the playing field by means of slowing down the processing speed of the algorithm at certain times. This is done by 'sleeping' the main thread of execution after each complete processing cycle through all the agents. The 'sleeping' is necessary to allow the user to see the animation on the screen. Without it, the speed at which the animation would occur would result in a blur on the screen and eliminate a great portion of the usefulness of this simulation.

After being initialized and started, the run method executes the actual applet. It cycles through the agents one at a time, updating their fitness functions and processing them for movement. A small percentage of the time (currently set at 5%) it 'kicks' the system and causes the agents to randomize their neighbors. This introduces a degree of stochasticity into the model. Every 100 cycles through the algorithm, the run method introduces a new agent into the program. This ensures that there is a constant pool of agents in the system and helps to offset the departure of agents that either separate or retire at the end of their career. The actual vessel which stores the individual agents is a Java class called a vector, which is essentially a linked list. Using a vector allows the program to reap all the benefits of linked lists (popping entities out of the middle and adding entities to either end) without the need to keep track of pointers. This alone makes Java an excellent programming language for handling this simulation. The run method will execute forever, continuously cycling through each agent in the system. Java's multithreading 'sleep' capabilities are employed at the end of each cycle so that the animation of the movement of the agents can be observed by the user.

The stop method terminates the main thread of execution, ending PICAS in an orderly manner. This method is never explicitly called but is activated when the program is terminated by the user.

2.4.3. SummaryStats Class

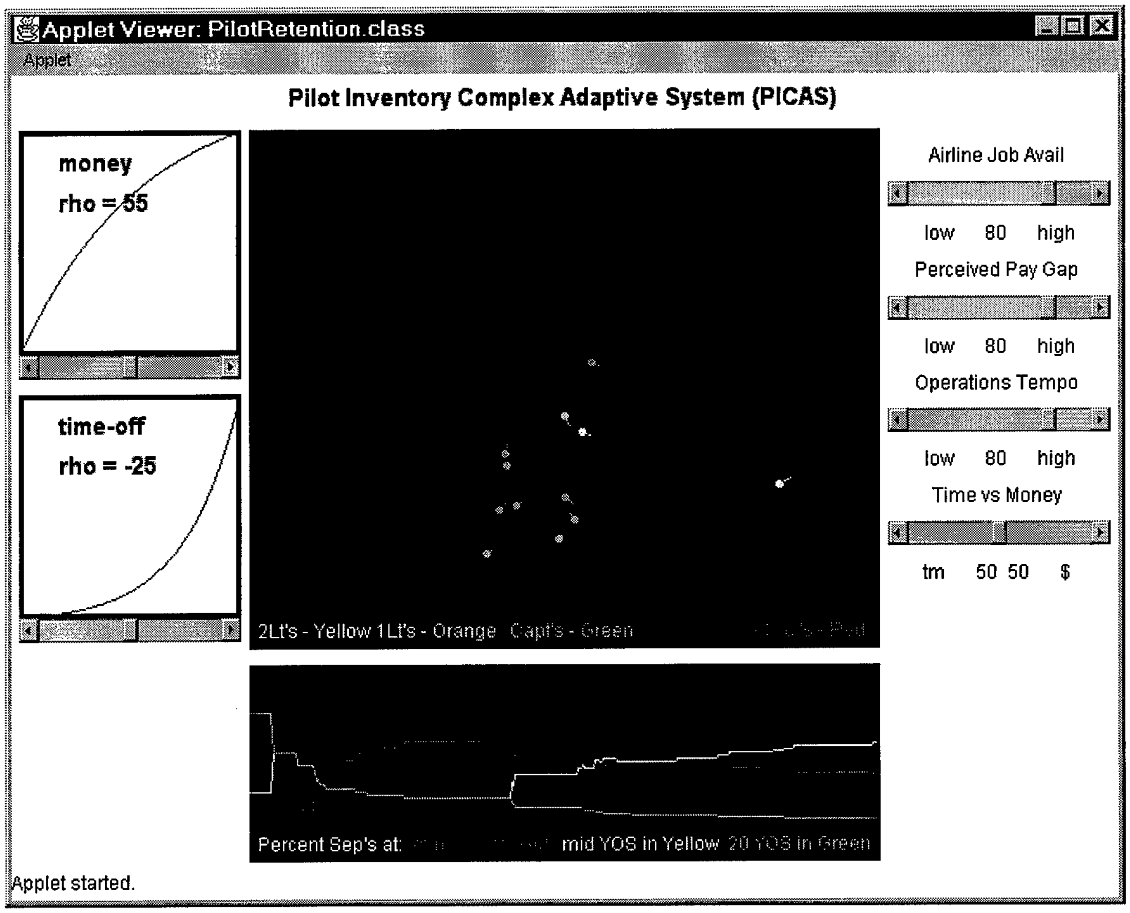
The SummaryStats class collects and displays quantitative information that is created during the execution of the simulation. This method allows the user to view model results and compare different model runs.

The statistics displayed are those regarding the separation time-frames of the agents. The percentage of agents separated at the minimum, middle, and end of career points are dynamically displayed in real time on a line graph at the bottom of the applet screen. Each agent that separates, regardless of his years-of-service, adds one to the denominator of these three statistics. Depending on the time-frame that the agent separates, the numerator of one of these three statistics is incremented by one. If the agent separates between 8 and 12 years-of-service, then the numerator of the minimum career point statistic is incremented by one. If the agent separates between 12 and 20 years-of-service, then the numerator of the middle career point statistic is incremented by one. Finally, if the agent separates at the 20 year point (and all agents are forced to separate at this point if they have not yet separated), the numerator of the end of career point statistic is incremented by one.

The separations occur in one of two ways. If any particular agent is present in the lower right-hand corner of the screen for 100 iterations through the entire system's algorithm, he is flagged as ready for separation. As soon as this agent meets his minimum service requirements, he is allowed to separate. The other method of separation occurs whenever an agent reaches the 20 years-of-service point. At this time he is automatically separated, analogous to retirement from the service.

2.4.4. Graphical User Interface (GUI)

The Graphical User Interface (GUI) allows the user to both view the animation and data output of the model as it runs and dynamically alter the characteristics of the model in real-time. The actual display is divided into 5 parts (see Figure 1: Screen Capture of PICAS Model), represented by Java's border layout manager. At the top


Applet Viewer: PilotRetention.class
Applet
2Lt's - Yellow 1Lt's - Orange Capt's - Green
Percent Sep's at:
mid YOS in Yellow 20 YOS in Green
Applet started.

Pilot Inventory Complex Adaptive System (PICAS)

money
rho = 55

time-off
rho = -25

Airline Job Avail
low 80 high

Perceived Pay Gap
low 80 high

Operations Tempo
low 80 high

Time vs Money
tm 50 50 $

Figure 1: Screen Capture of PICAS Model

(north position) is the title of the model. On the right (east position) are the environmental controls. On the left (west position) are the pilot controls. On the bottom (south position) is the display of the summary statistics. In the center (center position) is the pilot-agents' playing field where the animation of the pilot-agents conducting their decision process is displayed.

The dynamic control of the model is handled by the GUI's interface with the user. Included in this interface are controls for both the environmental parameters as well as the parameters of the pilots themselves. The user is able to alter one or all of these parameters and observe the effects immediately on the simulation in progress.

The environmental parameters are controlled by means of a series of slider bars, one for each of the environmental controls included in the model. The potential for employment as a commercial pilot is controlled by the slider bar called 'Airline Job Avail'. The 'Perceived Pay Gap' slider controls the (perceived) degree of monetary discrepancy between Air Force pilots and their commercial counterparts. We say 'perceived' to differentiate from any actual pay-gap data currently available. This also reinforces the notion that perceptions with regard to pay drive the pilots' behaviors, irregardless of the actual paygap. Lastly, the intensity of flying operations are controlled by the slider entitled 'Operations Tempo', a notional concept related to deployment demands and thus lack of time off or family time. All of these sliders can be altered from a low state (represented by a zero) to a high state (represented by 100). A bad environment is simulated by setting these parameters at high levels (say 80 or higher), while a good environment is represented by low settings (say 20 or lower). At the middle setting (around 50) the environment is considered average. The numbers, actually, do not mean anything explicitly to the user, but they do add a degree of scale to the model. The numbers allow the user to see exactly how much he is changing a certain parameter. Each environmental parameter can be changed independently of the other parameters and all effects are felt immediately by the simulation. This real-time interaction allows the user to observe the effects of his actions on the simulation without having to restart or rerun the entire model every time a parameter is altered.

The individual pilot-agent parameters are controlled by means of two utility curves, one representing money and the other representing time-off. These utility curves are controlled by a slider bar at the bottom of each curve (see Figure 2: Utility Curve Example). Sliding the bar in either direction increases or decreases the curvature of the

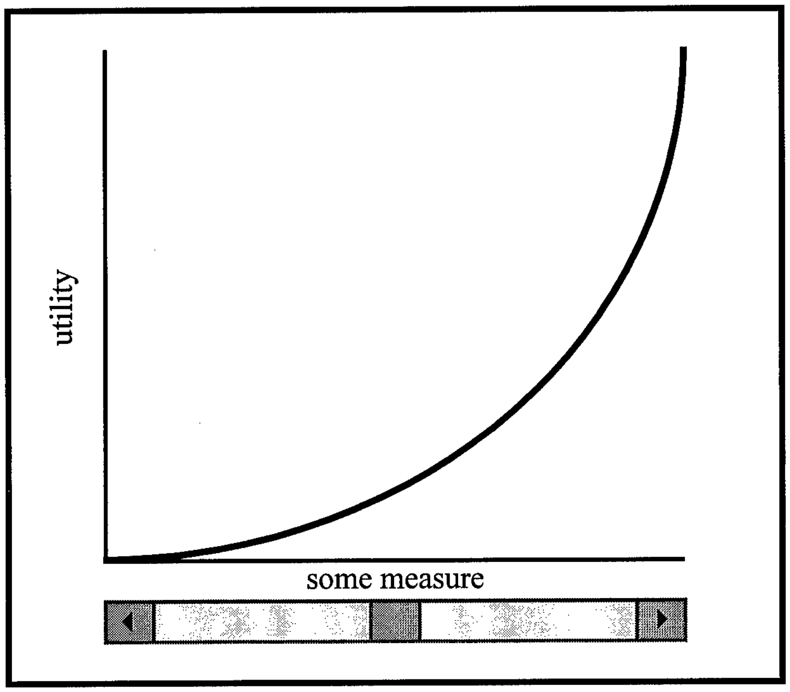


Figure 2: Utility Curve Example

entire curve and therefore changes the dynamics of the pilot's reaction to his environment. The utility curve idea is based on the fact that a certain degree of utility—measured on the Y axis—is derived by a given amount of a certain measurable quantity—depicted on the X axis. The slider bar allows the user to alter the shape of the curve making it either concave (bowed downwards towards the lower left as in Figure 2 above), convex (bowed upwards towards the upper right), or a straight line. Utility theory is used to approximate the pilots' attitudes towards service by capturing their satisfaction with regards to the money they make and the amount of time-off they have.

Two examples illustrate how to interpret these curves. Looking at the time-off curve, moving the slider bar left bows the curve upward and to the left, thus creating a convex utility curve. For such a curve, even a small X-axis input would result in a decent degree of utility. Therefore, even amidst a bad environment (represented by a high 'opstempo' setting which is in-turn converted to a small X-axis input on the utility curve) a good degree of satisfaction will be arrived at from a convex utility curve. For such a curve the environmental parameter ('opstempo' in this case) does not play a big part in determining the pilot's career aspirations. Conversely, moving the slider right bows the curve downward and to the right, thus creating a concave utility curve. A small X-axis input for such a curve results in a very low degree of utility. Therefore, given a bad environment, the pilot's concave utility curve will produce a low utility score, thus inducing the pilot towards separation. For concave utility curves, the environmental parameter plays a large part in answering the separation/retention question.

Coupled with the fourth slider bar below the three environmental slider bars, the user may dynamically alter both the shape of the pilots' utility curves and their weight with respect to each other. This allows the user to completely control the pilots' overall attitudes towards their intentions regarding separation and retention.

The controls in the model all work together to alter the fitness function of the pilot-agents within the system, therefore allowing them to adapt based on their personal attitudes (as captured by their utility curves) and the status of their environment (as captured by the environmental controls). The values from the 'Airline Job Avail' slider bar and the 'Perceived Pay Gap' slider bar are combined (by means of averaging) and then used as the X-axis inputs into the utility curve representing the pilot's attitude towards money. The 'Operations Tempo' slider bar value is used as an X-axis input into the 'time-off' utility curve. The utility values (measured on the Y-axis of each utility curve) are then combined to derive a new fitness function for each agent. This new fitness function is compared to the agent's current fitness function, and causes the current fitness function to adjust so that it coincides more with what the current environment represents. In this way, the agents 'learn' and adapt gradually to their environment, and eventually exploit it to fulfill their career aspirations. If the fitness function dips below a certain threshold, the pilot-agents will migrate towards the separation attractor and eventually leave the system. Conversely, if the fitness function remains at a high enough level, the pilot-agents will either be in an undecided state or migrate towards the retention attractor.

2.5. Verification and Validation

2.5.1. Verification of the Model

Verification of the PICAS model was accomplished throughout its development. All subroutines and methods were vigorously tested in stand-alone applications for accuracy before they were integrated into the entire model. Various intermediate results were derived and examined for accuracy. This helped to establish algorithms and code that mimicked the system's actual behavior.

Upon completion of the coding aspects of the model, numerous runs were conducted at various different parameter settings to ensure that the program handled all types of conditions. The animation presented by the model was also examined for realism and determined to be both intuitive and insightful.

Overall, verification was fairly straight forward due to the rather simple construction of complex adaptive systems. By their very nature, complex adaptive systems are comprised of agents that follow simple rules. The interaction of these agents, however, is what leads to complexity and emergent behavior that is nearly impossible to pre-program. The PICAS model follows this paradigm perfectly. The agents are subject to simple rules, but their apparently chaotic behavior on the screen is both complex and compelling to watch.

2.5.2. Validation of the Model

The PICAS model was designed as a descriptive model, not a prescriptive tool. As such, recreating past behavior is more important than the ability to predict the future. This complicates model validation. For the PICAS prototype, strict validity is not crucial, but initially valid behaviors are desirable. For our purposes, valid behaviors are deemed present if the emergent behaviors presented by the model suggest what history found to hold. Our two cases are pre-Desert Storm and post-Desert Storm. For each, we define the scenario and set the notional slider scales to those values providing the proper parameter settings.

Before the Gulf War, in the late 1980's, a draw-down was in effect throughout the Department of Defense. This meant that pilot retention was not a real issue, since pilots that separated on their own spared the Air Force the burden of forcing pilots out of the service in order to meet draw-down requirements. Setting the environmental parameters to reflect a poor environment (i.e., values around 85) and creating indifferent utility curves for pilots' attitudes (i.e., rhos of 0) would mimic this scenario and the PICAS model presents a definite degree of separations before retirement.

During Desert Storm, however, so many early separations would not have been beneficial, since the Air Force needed every trained pilot it could muster to gain air supremacy over the skies of Iraq and maintain the air bridge from the United States. Despite the continued economic prosperity in the United States (i.e., hiring value set at 85) and the increasing wage gap between the military and civilian aviation sectors (i.e., paygap set at 85), pilot retention was not adversely affected. This can be explained by the pilots' commitment to the Air Force core value of 'service before self' and can be modeled in PICAS by swinging the utility curves towards the right and creating convex functions (i.e., with rhos around 25). During the war, pilots attitudes towards money and time-off were strikingly positive. Even though Desert Storm forced many pilots to be separated from their families for months and subjected to incredibly high operations tempos (i.e., opstempo set at 95), pilots did not seem to mind because they were fulfilled by the requirements of their jobs. This positive attitude is reflected in the PICAS model by inducing a very convex nature to the pilots' utility curves. With highly convex utility curves, even if X-axis inputs are relatively small, a great degree of utility can still be achieved. This allowed pilots' fitness functions to remain high enough to avoid separation, despite the fact that the environment was quite unappealing towards them.

As Desert Storm came to an end, however, the environment gave no indication of reverting back to a more pro-retention position. The commercial sector, already appealing during the war, remained so after the war. Airlines were hiring and paying well; reflected by setting hiring and paygap at 85 in the PICAS model. The Air Force, however, was not doing much to increase pilot pay, and the operations tempo remained incredibly high (i.e., set at 85). With no war to rally behind, pilot's attitudes got progressively worse; their utility curves went from convex to indifferent (or straight-lined) to concave (i.e., negative rhos). As such, pilots were increasing their propensity to separate from the service and enter commercial aviation. By altering the utility curves in the PICAS model to make them more concave, this behavior emerges. Today, the Air Force is in a similar predicament. Commercial aviation is thriving and continues to advertise better jobs with more pay. The operations tempo is still at record highs, and pilots have no real-world missions to revive their esprit-de-corps. As such, attitudes towards money and time-off are getting more pessimistic (as represented by concave utility curves) and more pilots are separating earlier in their careers to remove themselves from the poor environment being created in the Air Force.

2.6. Analysis Methods Using Complex Adaptive Systems

2.6.1. Prescription versus Prediction

Among the biggest criticisms of complex adaptive systems theory and artificial life approaches to agent-based modeling is the fact that repeatability of experiments is very difficult. This leads to problems with developing confidence intervals for experiments, hence detracting greatly from the predictive power of such a model. Complex adaptive systems theory advocates, however, point out that complex adaptive systems are best used in models for their prescriptive abilities to provide insight.

While the notion of using models for prescriptive purposes has been around for a long time, modern simulation practitioners have under-emphasized this side of modeling. Many computer models in use today try to predict the future. However, many models used throughout human history were aimed at describing the workings of a system, not making predictions about it. Any biology student knows the benefits of using a model of the human body to help understand all its organs and bodily systems. The model does not provide predictions for the outcome of medical procedures. Examining and understanding the model, though, does lead to a profound understanding of the entire system, which in turn can be used to gain insight into the possible ramifications of certain drugs and/or medical procedures. So, although the model does not predict anything by itself, understanding the model helps the user deduce future events by assimilating the knowledge he has observed and processing it in a logical manner.

In the same way, the pilot retention issue will not be solved by the results obtained from a CAS model, but insights gained from it could provide policy makers a better understanding of the inner workings of the system. If the model can be built in such a way as to mimic reality with a high degree of accuracy, then the model can be examined and the real system better understood. By understanding the true system, the model user gains insight into the possible results of certain policy decisions. This information can help guide policy actions by decision makers.

Although they may lack specific prescriptive capabilities, CAS-based models allow their users to determine some aspects about the predictive power of their models. For one thing, because complex adaptive systems theory is based on chaos theory, these models generally contain certain basins of attraction that seem to cause emergent behavior to gravitate towards small zones of feasibility. Therefore, the user is able to determine with relative certainty that the end result of the model's execution will lie in some basin of attraction. This means that although the user might not know which basin of attraction the model results will drive to, they know it will drive to some basin of attraction. If the basins of attraction are smaller than the entire feasible space that the agents can occupy, then knowledge of these basins is very informative with regards to predictive capabilities. Instead of knowing where the system is in the future, the user can tell where the system won't be in the future. This can focus attention and analysis on those areas that contain the attractors, and may lead to the discovery of scenarios and parameter settings that induce results into the preferred basins of attraction.

In the pilot retention model, this information can be very important. Looking at the playing field, there are basically three basins of attraction: the undecided basin, the retention basin, and the separation basin. During model execution, pilot-agents move towards one of these attractors. Policy decisions either induce pilots to separate from the Air Force or persuade them to stay in the service. Those decisions that result in a clear-cut decision one way or the other are easy to understand, but those that result in fairly even splits are the ones that would require further research. Again, the model user can now concentrate his efforts on exploring those options that seem to produce the ambivalent results. The model has helped focus on the important issues.

2.6.2. Exploratory Modeling

Exploratory modeling involves creating a database of models (or individual model runs) that can then be referenced once a given decision must be addressed. By creating models that run the entire gamut of the system's feasible space, the database created can be used to come up with a response function for all the parameters of the system. This would allow a user to determine if there are any local extrema within the system's feasible space. Such information could be used to help maximize the decisions involved in making the system run at peak performance.

For the pilot retention problem, exploratory modeling helps determine which policies yield the greatest benefit with regards to pilot retention. This allows the model user to narrow in on the models that warrant further investigation. The goal is spending less time on the modeling exercise and more time refining those policy initiatives that show promise.

2.7. Model Results

2.7.1. Model Results from Various Parameter Settings

To analyze the results produced by the model, a series of experiments were conducted at various different parameter levels. In an effort to cover a wide range of parameter values, an experimental design was created varying each parameter between a relatively high value and a relatively low value (see Table 1: Experimental Design for Analysis of Model Results). Specifically, for the three environmental parameters, each was varied between either 20 (representing a good environment) and 80 (representing a bad environment). The pilot attitudes were varied by means of changing the rho (ρ) value of their utility curves. A rho of –25 represents a concave utility curve, meaning a high X-axis input is required to create a good degree of utility. Conversely, a rho of 25 represents a convex utility curve, which does not require a high X-axis input to derive a decent level of utility. Lastly, a rho of 0 represents a straight line, or in utility theory, indifference with respect to the utility arrived at by any given X-axis input. For a straight line utility curve, the amount of utility derived from a given input is linearly related to that input. Table 1 summarizes the various settings depicted in the graphical results below. Each number in the table corresponds to the three environmental parameter

Table 1: Experimental Design for Analysis of Model Results

time	money	opstempo = 20				opstempo = 80			
		paygap = 20		paygap = 80		paygap = 20		paygap = 80	
		hire = 20	hire = 80	hire = 20	hire = 80	hire = 20	hire = 80	hire = 20	hire = 80
-25	-25	1	10	19	28	37	46	55	64
	0	2	11	20	29	38	47	56	65
	25	3	12	21	30	39	48	57	66
0	-25	4	13	22	31	40	49	58	67
	0	5	14	23	32	41	50	59	68
	25	6	15	24	33	42	51	60	69
25	-25	7	16	25	34	43	52	61	70
	0	8	17	26	35	44	53	62	71
	25	9	18	27	36	45	54	63	72

settings in its column and the two pilot attitude settings in its row. As such, all 72 numbers in the table represent a full factorial design of every permutation of the two levels (20 and 80) represented by the three environmental parameters and the three levels (-25, 0, and 25) represented by the two pilot attitude parameters. These numbers are referenced by the graphs to help determine the settings of the parameters that lead to the behavior displayed on each graph.

For each of the experimental design settings conducted here, 5 runs were completed and the results were aggregated into an average. This data was then graphed for each type of separation: at minimum years-of-service, at the middle career point, and at the end of career point.

Separations at the minimum years-of-service point (see Figure 3: Separations at the Minimum Years-Of-Service Point) showed two distinctive patterns in the separation

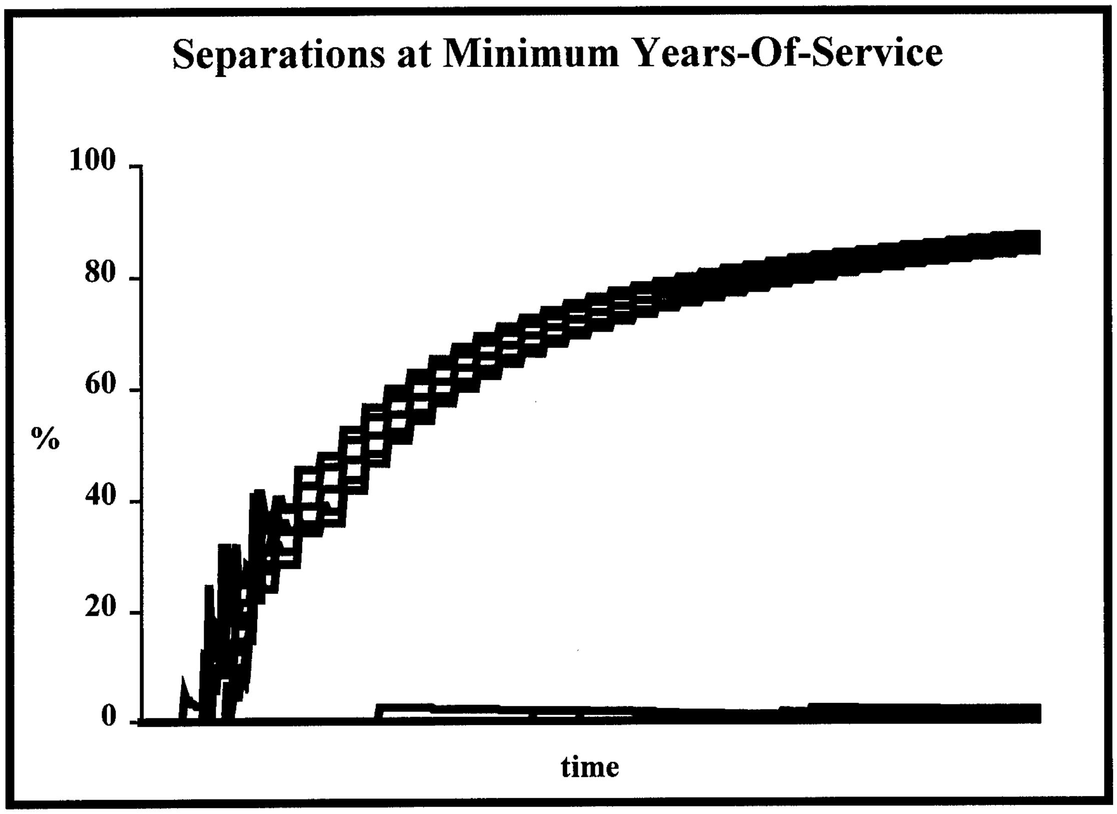


Figure 3: Separations at the Minimum Years-Of-Service Point

trends of the pilot-agents. The upper group of lines on the graph represent runs number 46, 49, 55, 58, 64, 65, 67. For these runs, according to Table 1, either the environment was very bad (i.e. two or more environmental controls were set at 80) or the pilot attitudes were poor (i.e. at least one utility curve was concave). The rest of the runs resulted in either very low separations (less than 5% of the total separations recorded throughout the run) or none at all. In either case, the emergent behavior resulted in the pilot-agents not separating from the service.

Separations at the middle career point (see Figure 4: Separations at the Middle Career Point) also showed two distinctive separation trends. Here again, the upper group

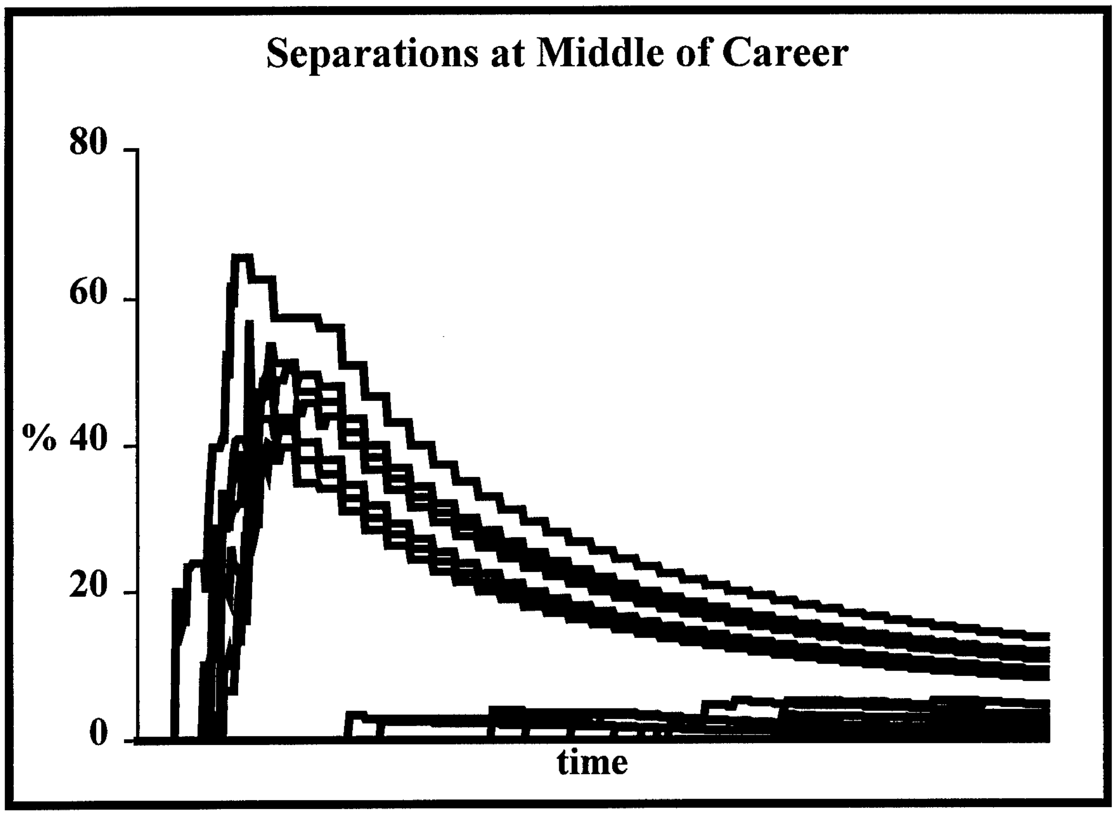


Figure 4: Separations at the Middle Career Point

of lines on the graph represent runs number 46, 49, 55, 58, 64, 65, 67. As above, these runs, according to Table 1, represent either a very bad environment (i.e. two or more environmental controls were set at 80) or poor pilot attitudes (i.e. at least one utility curve was concave). The spike that occurs near the beginnings of these runs is interesting. When originally created, the model starts with a random set of pilot-agents. These original pilots have a uniformly distributed age represented by their years-of-service. Therefore, if the environment and/or pilot attitudes are such that separation occurs, all eligible agents will separate. Naturally, this means that some agents will be below 12

YOS, and therefore counted as separating at the minimum YOS point, while others will be above 12 YOS but not yet retirement eligible. When they separate, they will be counted as separating at the middle of their career. With a poor environment or pilot attitude, though, as new agents are introduced into the model, they will all separate as soon as they can, thus increasing the minimum YOS statistic's numerator (and therefore its resultant percentage) while only the denominator for the middle career separators increases (thus lowering its resultant percentage). As time goes on and more agents separate as soon as they can, the middle career point statistic continues to decrease, hence the downward sloping nature of these curves. Exactly as with the minimum separators, the rest of the runs resulted in either very low separations (less than 5% of the total separations recorded throughout the run) or none at all.

Lastly, separations at the end of career point (see Figure 5: Separations at the End of Career Point) showed the same two distinctive separation trends. The lower group of lines on the graph, representing runs number 46, 49, 55, 58, 64, 65, 67, correspond to either a very bad environment (i.e. two or more environmental controls were set at 80) or poor pilot attitudes (i.e. at least one utility curve was concave). These lines show that retention for the entire pilots' careers goes down drastically because the system is driving the pilot-agents to separate early. Initially, due to the nature of the model, an older pilot (one with nearly 20 YOS) will retire before he is affected by the environment. The step-wise progression of the lines towards the right of the graph are the result of averaging the five runs into one data point. These steps occur at exactly 20%, 40%, 60%, and 80%. If for any given run the first agent separates at the 20 year point due to his age, then the first data point graphed by the SummaryStats class will be a green line at 100%. This is because the first and only data point until that time is an end of career separator. However, in the other four runs used to arrive at the aggregate data point graphed below,

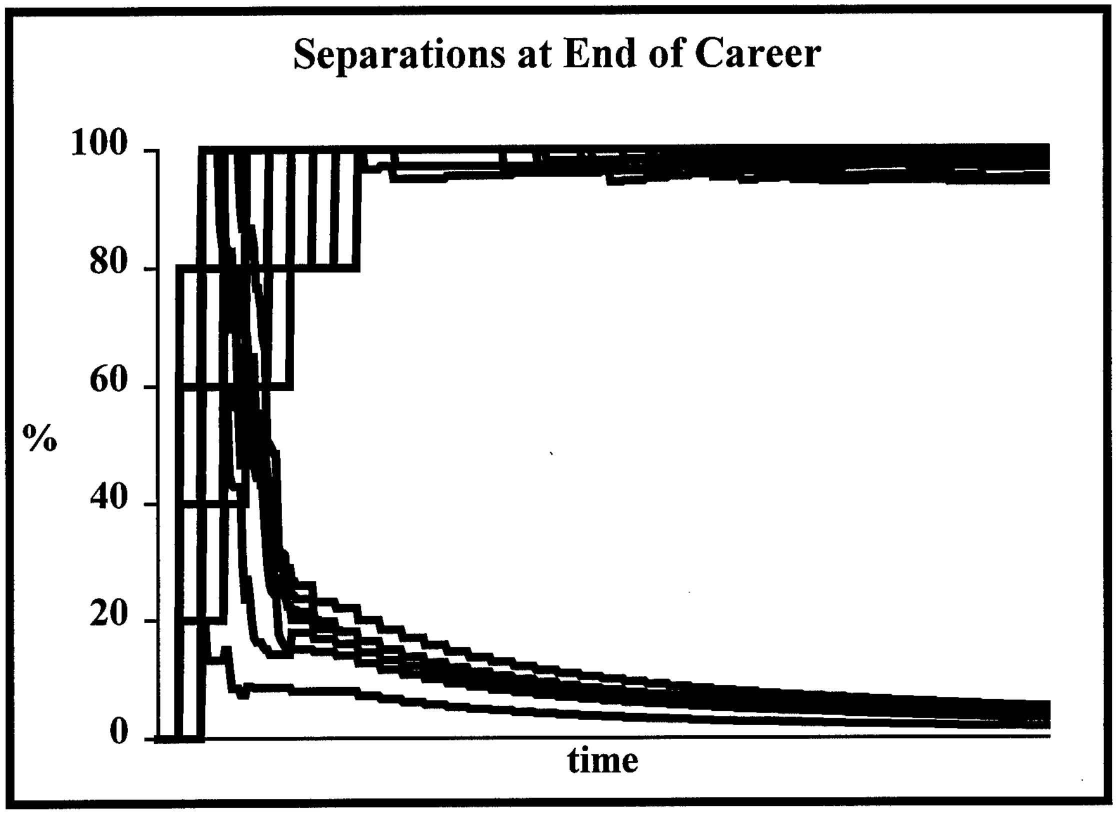


Figure 5: Separations at the End of Career Point

there may or may not have been a separation at this time yet. Therefore, when the data is averaged, the step-wise behavior will occur. Again, exactly as shown before, the rest of the runs resulted in very high retention (95% or better) until the end of career point. This result confirms our intuition since parameter settings maintaining a good environment and/or a positive pilot attitudes should eliminate the early separation of pilots.

2.7.2. Response Surface Methodology Results

To get a better understanding of the interactions of the various parameters in the model and their effects on the pilot-agents' propensities towards separation and retention, a regression analysis was conducted using response surface methodology (RSM). Using RSM allowed us to determine the shape of the response surface created when various parameter settings were used. It also allowed us to examine the interactions between various parameter settings and determine if any particular type of environment and/or pilot attitude combination produces particularly interesting results that would warrant further investigation.

To start off the entire process, an experimental design had to be created. A central composite design with face-centered axial points was determined to be the best design for our analysis. The coding scheme used to create this central composite design required that each of the five parameters be varied above and below its center point by an equal amount (represented by +1 and -1 in the coded space). This resulted in the coding schemes depicted in Table 2 below. In addition to the 32 full factorial design points

Table 2: Coding Schemes for RSM Analysis

parameter	-1 coded to	0 coded to	+1 coded to
money	-25	0	25
time	-25	0	25
hiring	20	0	80
paygap	20	0	80
opstempo	20	0	80

required to analyze this model, 10 axial points and 10 center points were needed to examine curvature. Due to the limited feasible region covered by the parameters in this model, face-centered axial points were used to test for curvature. These face-centered points were therefore coded as +/-1. Each design point was arrived at by conducting 5 runs and then averaging the results. The data point used was the last observation of each run (i.e. the 400th observation). This resulted in three sets of data, one for the percentage of pilots separating at the minimum YOS point (min), one for the percentage separating at the middle career point (mid), and finally one for the percentage separating at the end of career point (end).

The 52 design points were analyzed using SAS JMP's fit model routine. For each set of points, a stepwise regression was conducted using a 0.05 probability to enter or leave the model. Based on the effects suggested by each respective stepwise regression, a final regression model was run for each set of data. Upon examination of the residuals, however, SAS JMP indicated that the normality assumption was violated for all three models. In each case, the W statistic from the Shapiro-Wilk test indicated that the distribution was not normally distributed (see Table 3: Normality Test Results For Regression Analysis).

Table 3: Normality Test Results For Regression Analysis

	W	p-value
minimum career separators	0.797831	<.0001
middle of career separators	0.945112	0.0301
end of career separators	0.842189	<.0001

Due to the lack of normality of the residuals, statistical inferences concerning the significance of effects could not be made based on the p-values given using standard linear regression methods. Therefore, the Kruskal-Wallis (KW) test was used to analyze the data. Since the KW test is a non-parametric test, there is no need to ensure that the residuals are normally distributed. This allowed us to determine if there were any statistically significant effects based on separation percentages as a result of the parameters settings we investigated. Table 4: Kruskal-Wallace Test Results, presents a summary of the information obtained using the Kruskal-Wallace test. Each effect was tested to determine whether or not it was statistically significant in the overall model. Table 4 presents the KW test-statistics and resultant p-values for each of the main effects and all possible two-factor interaction effects in all three scenarios (minimum career separators, middle of career separators, and end of career separators).

Table 4: Kruskal-Wallace Test Results

	min		mid		end	
term	KW-stat	p-value	KW-stat	p-value	KW-stat	p-value
MEAN	0	0	0	0	0	0
money	2.5081	0.2854	4.8377	0.0890	4.5462	0.1030
time	7.8918	0.0193	7.0392	0.0296	7.0803	0.0290
hiring	2.1805	0.3361	2.3688	0.3059	2.3374	0.3108
paygap	3.8674	0.1446	1.6872	0.4302	2.4793	0.2895
opstempo	2.5081	0.2854	6.1302	0.0466	5.5882	0.0612
money*money	1.3468	0.2458	1.9404	0.1636	1.9667	0.1608
time*money	2.3256	0.3126	1.1829	0.5535	0.9623	0.6181
time*time	3.1037	0.0781	0.4557	0.4996	0.8280	0.3629
hiring*money	1.9839	0.3709	1.4152	0.4928	1.5673	0.4567
hiring*time	2.9342	0.2306	0.6546	0.7209	0.8780	0.6447
hiring*hiring	1.3468	0.2458	0.1176	0.7317	0.2022	0.6530
paygap*money	2.9675	0.2268	0.8071	0.6679	0.8714	0.6468
paygap*time	1.9868	0.3703	0.7086	0.7016	0.9076	0.6352
paygap*hiring	3.0014	0.2230	1.2422	0.5373	1.1519	0.5622
paygap*paygap	1.3468	0.2458	0.5788	0.4468	0.6795	0.4098
opstempo*money	2.0213	0.3640	0.8531	0.6528	1.0939	0.5787
opstempo*time	9.8608	0.0072	3.9923	0.1359	5.1266	0.0770
opstempo*hiring	2.0213	0.3640	1.6098	0.4471	1.5673	0.4567
opstempo*paygap	2.6880	0.2608	0.8294	0.6605	1.1829	0.5535
opstempo*opstempo	1.3468	0.2458	0.9508	0.3295	1.0562	0.3041

Using a 0.05 level of significance, each model was examined to determine which effects were statistically significant in explaining the variation within each data set. For all three data sets, the 'time' variable was determined to be statistically significant. Added to this, the 'opstempo*time' interaction effect was statistically significant in the minimum career separators model and the 'opstempo' effect was statistically significant in the middle of career separators model.

Using this information, each regression model was then re-run with only its statistically significant terms included. The effects' estimates and the model's statistics are presented in Table 5 below:

Table 5: RSM Results of Statistically Significant Model Effects

term	min	mid	end
MEAN	5.1500	1.1719	93.6783
time	-7.6350	-1.0685	8.7032
opstempo	N/A	1.2274	N/A
opstempo*time	-8.2400	N/A	N/A
R^2	0.1986	0.2661	0.0989
adjusted R^2	0.1659	0.2361	0.0809
Shapiro-Wilk W statistic	0.6147	0.7582	0.5368
Shapiro-Wilk p-value	<.0001	<.0001	0.0000

When the residuals for these models were examined, again, it was determined that the error terms were not normally distributed (see Shapiro-Wilk W statistics in Table 5). The lack of normality comes as no surprise; the data represented here is primarily bi-modal—a pilot either separates or stays in. After 400 observations, the resultant percentage of separations is either very high or very low with little in-between. Such bi-modal data does not lend itself very well to conventional regression analysis, but prescriptive inferences from the analysis are still obtainable.

Looking at the signs of the parameter estimates presented in Table 5 is another source of verification for the results presented by the PICAS model. For the percentage of separations at the minimum YOS point and the middle career point, the sign of the time parameter estimate is negative. This means that a negative setting for this parameter leads to a higher percentage of separations. A negative setting for the time parameter corresponds to a concave utility curve, requiring a greater X-axis input to acquire a decent degree of utility. This agrees completely with our intuition. As pilot attitudes get worse, the separation issue gets harder to deal with. The same parameter has an estimate with a positive sign for the percentage of separations at the end of career point. This means that a positive setting leads to a higher percentage of separations at the end of career point. Since a positive time setting represents a convex utility curve which does not require a high X-axis input to achieve a high degree of utility, this again agrees with our intuition. If pilot attitudes are positive, then most will remain in the service until they separate at retirement.

Looking at the estimates for the opstempo environmental parameter settings, the same kind of results are apparent. For the data representing pilots separating at the middle of career point, the estimate for opstempo is positive. This means that a higher setting will result in more separations. Since a higher setting corresponds to a harsher environment, we are comforted in the intuitive appeal of this result.

The sign of the interaction term deemed statistically significant for the minimum career separators model, although somewhat harder to explain intuitively, does agree mathematically with the above discussion. Basically, the opstempo*time interaction term's sign is just the multiplicative combination of the signs of the opstempo estimate (which would be 7.5747 if opstempo were included in this model—it isn't because it wasn't deemed statistically significant by itself) and the time estimate. Other than this, this term does not provide any real insight into the issue, and attempting any further explanation is dangerous.

As an aside, it is interesting to note that the hiring and paygap environmental parameters and the money pilot attitude parameter are not present in any of the models. The cause of this lies in the fact that these two environmental parameters are actually combined to produce the X-axis input into the money utility curve. As such, the effects of all three parameters are diluted by the model, and the other effects surpass them in significance. This again confirms our intuition about how the PICAS model was built to perform.

2.7.3. Three Dimensional Graphical Analysis of Results

In order to provide a better graphical representation of the results presented by the PICAS model, we endeavored to create a surface plot encompassing all five parameters for each separation statistic. The problem, however, was that we needed to aggregate our five independent variables into the two dimensions required to create such a surface plot. Since the PICAS model provides two basic means for control—altering pilot attitudes or altering the system's environment—we felt this was the natural way to go.

Pilot attitudes, modeled by the money and time-off utility curves, were represented by summing the rho settings for the combinations of utility curves used throughout the data analysis conducted in sections 2.7.1 and 2.7.2. This resulted in an overall pilot attitude parameter with values of -50, -25, 0, 25, and 50. The overall environment was depicted in a similar manner. With three separate dimensions representing airline hiring practices, the perceived paygap, and the current operations tempo, the overall environmental parameter was also created using summation. In this case, the combinations of the three environmental dimensions used throughout analysis of the PICAS model resulted in overall environmental parameter settings of 60, 120, 150, 180, and 240. These two parameters, which captured all the data in the five parameters of the PICAS model, were then used as the X and Y axes on a surface plot for each set of separation data (minimum YOS, middle career point, and end of career point).

Looking at the surface plot of the separations at the minimum YOS point, no surprises are apparent (see Figure 6: Surface Plot of Minimum YOS Point Separators).

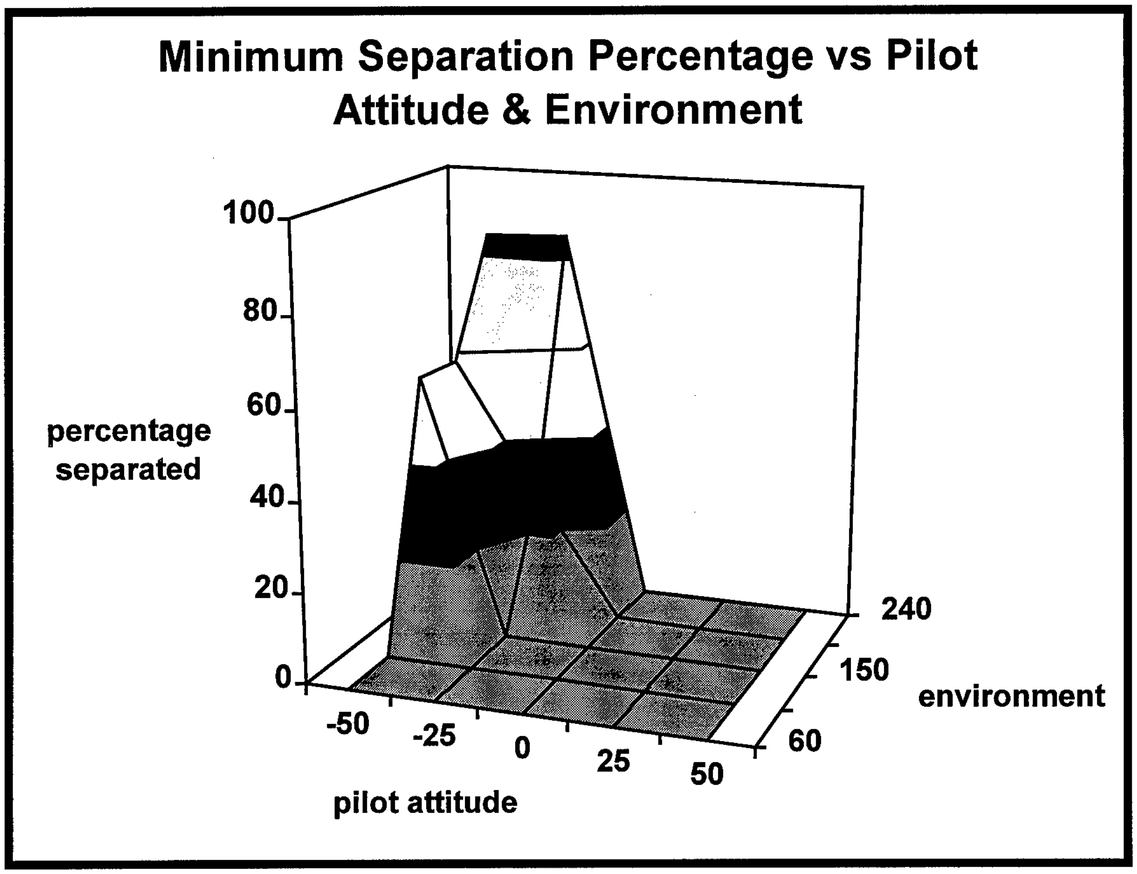


Figure 6: Surface Plot of Minimum YOS Point Separators

As can be expected, only an average to poor environment (settings of 150 or higher) coupled with negative pilot attitudes (rhos of -25 or lower) results in significant separations early in a pilot's career. This confirms with reality and also provides policy makers an important piece of information. If either the pilot's attitudes or the environment can be made better, then early separation could be reduced significantly. Also, the graph indicates that the environment seems to affect separation more than pilot attitudes. Even at an average pilot attitude (rhos summing to 0), separation is not an issue. But, for average environmental settings (settings of 150), if the pilot attitude is really poor (rhos summing to –50), then separation becomes an issue. Based on this information, it may be prudent for policy makers to concentrate on improving the pilots' environment rather than improving pilots' attitudes.

The surface plot of the separations at the middle career point is a bit more interesting (see Figure 7: Surface Plot of Middle of Career Separators). Just like the surface plot of the separations at the minimum YOS point (in Figure 6 above), this surface plot shows that separations are the highest whenever both the environment and the pilots' attitudes are poor. However, the separations at the middle of career point also increase if either one of these parameters is at its worst, regardless of the setting of the other parameter. If the environment is at its worst setting, corresponding to a value of 240, then even if the pilots' attitudes are at their best, corresponding to a value of 50, there are a significant amount of separations at the middle career point. In the same manner, if the pilots' attitudes are at their worst setting (-50), and the environment is at its best setting (60), there is still a rise in the percentage of separations at the middle career point. Comparing these two extremes, however, it is very apparent that the latter is not as severe as the former. A very poor environment coupled with excellent pilot attitudes results in a much greater separation percentage than very poor pilot attitudes coupled with an outstanding environment. This information is very valuable to policy makers. It indicates that given a choice between which issue to attack first, the best course of action would be to concentrate on making the environment more favorable instead of trying to improve the pilots' attitudes. Added to this, this conforms to the results of the surface

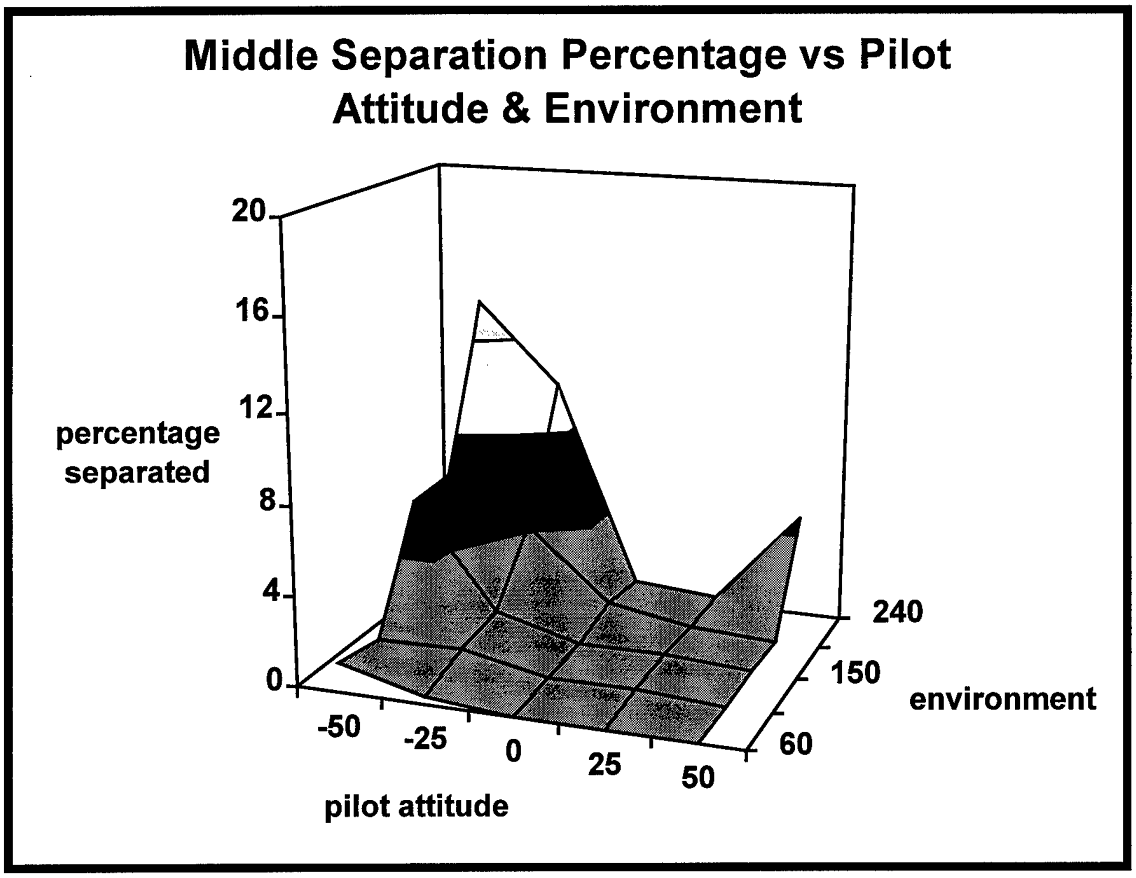


Figure 7: Surface Plot of Middle of Career Separators

plot of minimum YOS separators presented in Figure 6 above.

The last surface plot depicts the percentage of separations that occur at the end of a pilot's career (see Figure 8: Surface Plot of End of Career Separators). This surface plot represents the percentage of pilots that remain in the Air Force until retirement. Compared to the other two surface plots, this graph is a mirror image of the sum of the data depicted on the previous two plots. As expected, this plot shows that pilots remain primarily career minded unless both the environment and their attitudes become very poor. Retention until retirement also dips a bit if either the environment is at its poorest

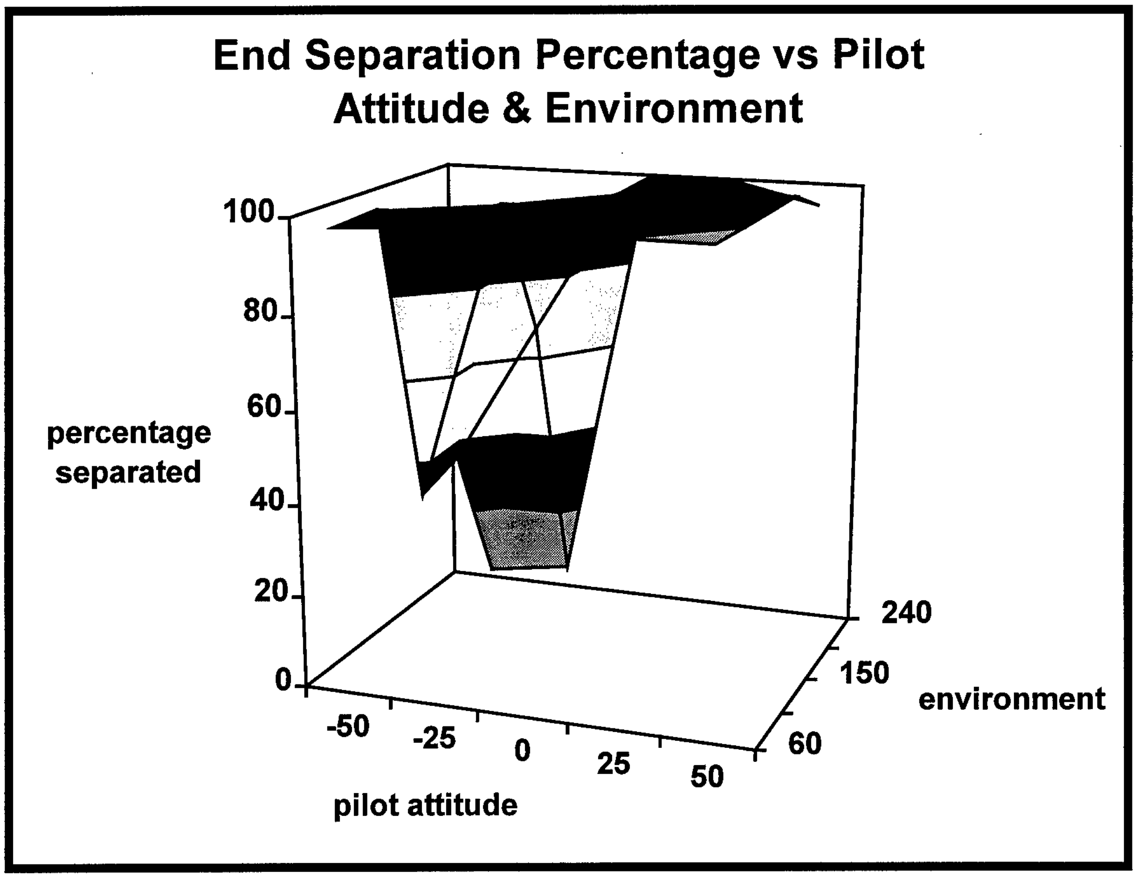


Figure 8: Surface Plot of End of Career Separators

level (regardless of pilots' attitudes) or the pilots' attitudes are at their poorest level (regardless of the environment). The dip, however, is greater if the environment is poor while pilots' attitudes remain positive. Once again, this implies that decision makers should concentrate more on enhancing the environment to induce retention, rather than trying to alter individual attitudes.

2.8. Results and Conclusions

Complex adaptive systems are an excellent way to model complicated behavior using autonomous agents following simple rules. The goal of all complex adaptive system simulations is to spot trends in the system's emergent behavior. Since the interactions between the agents in the simulation are characteristic of a chaotic system, one in which the underlying order appears random on the surface, the ability to make short-term predictions is all but non-existent. However, chaotic systems do display long term trends, and it is for this reason that complex adaptive systems are useful. The goal is to take advantage of the system's tendencies toward self-organization and look at the emergent behavior inherent in the system's local chaotic actions and interactions. In the long run, we may not know exactly where the system is going to be, but we at least have a notion of the state space involved. As Holland points out, the emergent phenomena in generated complex adaptive systems are typically persistent patterns with changing components (Holland, 1998:225). For our purposes, as new pilots enter the model and old ones leave (either via retirement or separation) the model's behavior will still tend to be the same because the system in the aggregate is not changing. The system is in steady state, and the turnover of a few agents will not significantly alter the system-wide behavior.

The persistent patterns that emerge can usually be shown to satisfy macro-laws. This means that the model may suggest policies useful in the entire Air Force, not just the pilot community. However, a model based on complex adaptive systems theory should be built for the entire Air Force personnel system and examined for the behavior that emerges from it.

The use of the computer to model the pilot inventory problem alleviates the need to experiment on the actual force. As Holland points out, in this manner:

> Computer-based models provide a half-way house between theory and experiment. Computer-based models are not experiments in the usual sense because they do not directly manipulate the world being modeled. Nevertheless, as the model is executed, patterns and symmetries will typically show up in the ongoing action… Such emergent characteristics can suggest experimental designs for real situations. (Holland, 1998:119-120)

This means that by using complex adaptive systems to model the pilot inventory problem, the Air Force leadership has a way to examine future program policies without harming the current force. If the computer simulation suggests favorable results, then confirmatory experiments on the actual force could be considered. This would save both time and money for planners in the Air Force and avoid costly embarrassments due to unforeseen circumstances or responses to policy initiatives.

One must not forget that computer models merely guide our decision making process. In this manner, they tend to serve two roles. In one respect, they "provide a rigorous demonstration that something is possible" (Holland, 1998:241). On the other hand, they "suggest ideas about a complex situation, suggesting where to look for critical phenomena, points of control, and the like" (Holland, 1998:241). A well-designed model provides insight into the system it represents. It allows the user to test out new ideas without risk of harm to the actual system. It enables the user to examine different paths of possibilities that could not be explored in the real system. Using a complex adaptive systems theory approach to model building, it is hoped that the pilot inventory problem in the Air Force may benefit from some of these insights. Future developments using such an approach to personnel modeling might even overcome the long-term predictive limitations of current models and allow us to better manage our inventory of personnel in the Air Force.

CHAPTER 3

3.1. Further Extensions of this Research Effort

There are many areas of further research on this topic. For the current model, more detail could be included in the actual definition of the agents that comprise the simulation. For instance, one could add more parameters aimed at differentiating between individual agents. These added parameters would help increase the reality of the entire model by adding more individuality to each agent, hence adding more legitimacy to the model's results. Along with the added parameters in the individual agents, more environmental controls could be added as a means to increase the model user's ability to alter policies and procedures that could effect the model's outcome.

The graphical output created by the model could be improved. Java's new 3D API might add some very interesting animation capabilities to the model's current animation sequence. This would help with the model's presentation and give the model more appeal to the average observer. The 3D API might also help make the summary statistics more informative.

Expanding the model beyond the pilot inventory arena is another area for improvement. By opening up the model to include other functional areas and/or the entire Air Force personnel system, the model might be able to attack some larger scale problems currently facing Air Force personnel analysts. Although keeping track of thousands of agents might be possible on a computer, the animation would get very busy. Therefore, some degree of aggregation would be necessary if such a large model were pursued. Perhaps, instead of representing individual pilots, each agent could be a representation of each rank within a given Air Force Specialty Code (AFSC). This would allow the model user to see which AFSCs are more prone to retention problems and which are not. Again, for this to work the parameters that make up the agents would have to be greatly increased in order to include all the necessary information pertinent to each particular AFSC.

3.2. Additional Areas of Study using Complex Adaptive Systems Theory

Complex adaptive systems theory is an incredibly fertile area of study. The paradigm shift towards agent based models occurring in the simulation community would definitely benefit from further research into complex adaptive systems and the whole artificial life ideal. There is ample opportunity to examine both the theory and the application of these cutting edge 'new sciences'.

From the complex adaptive systems theory viewpoint, there are many legacy models that could very easily be converted to agent based models employing a complex adaptive system. The results from these new CAS models could then be verified and validated by being compared to their older counterparts. This would help to further legitimize the notion of using complex adaptive systems theory in the future of simulation and modeling.

Artificial life is also another very fertile area of study. Just defining what 'life' is could involve years of study. Recreating life in the computer, as opposed to the test tube, could be a very rewarding experience for any budding Operations Research scientist. Creating agent based models that exhibit life-like qualities could lead to some very interesting simulation exercises. The ability to recreate life-like behavior in a simulation would greatly enhance the prescriptive and predictive capabilities of computer models and would add much credibility to the entire modeling and simulation movement.

Cooperation with other sciences, such as biology, chemistry, physics, and psychology, to name a few, would also help increase the use of agent based models. By attacking long standing problems in these other sciences using a complex adaptive systems theory approach, the agent-based paradigm could win allies in areas that simulation never really dared to tread. In the end, the sky is the limit. There really is nothing that cannot be modeled as a complex adaptive system. Even 'dumb' agents can be created with little or no adaptive abilities. The key is to use agents, build from the bottom-up, and let the system's emergent behavior be what the model produces. The surprises that such agent-based, bottom-up models could produce may indeed shock some, but in the end, the results will help shape the future of simulation and modeling as we know it.

3.3. Concluding Remarks

A mere twenty years ago, scientists believed that simple systems behaved in simple ways and that complex behavior implied complex systems at work. The emergence of chaos theory and complex adaptive systems theory, however, has changed all of this. As Gleick shows:

> Now all that has changed. In the intervening twenty years, physicists, mathematicians, biologists, and astronomers have created an alternative set of ideas. Simple systems give rise to complex behavior. Complex systems give rise to simple behavior. And most important, the laws of complexity hold universally, caring not at all for the details of a system's constituent atoms. (Gleick, 1987:304)

Now complex systems can be modeled by simple agents, and simple models can lead to complex behavior. Chaos theory has highlighted the 'gray' space between deterministic behavior and completely random stochastic behavior. Chaos theory has lead to a revolution in the sciences, one that has opened up doors to problems we never even imagined solvable.

As for the scientists who pioneered the study of chaos, Gleick claims that:

> Chaos was the set of ideas persuading all these scientists that they were members of a shared enterprise. Physicist or biologist or mathematician, they believed that simple, deterministic systems could breed complexity; that systems too complex for traditional mathematics could yet obey simple laws; and that, whatever their particular field, their task was to understand complexity itself. (Gleick, 1987:307)

The lesson learned here was one of simplification—simplify the model and let the model surprise you with its results. Creating models that are dynamic in nature allows us once and for all to see creation at work. By exploiting the notion of complex adaptive systems theory, we are suddenly able to create simple models that can be used to explain the most complex behaviors in the natural world. What we have found is that "nature forms patterns. Some are orderly in space but disorderly in time, others orderly in time but disorderly in space. Some patterns are fractal, exhibiting structures self-similar in scale. Others give rise to steady states or oscillating ones" (Gleick, 1987:308). In the end, though, these patterns give us insight into behaviors we never saw before. We gain knowledge about our natural environment, and that knowledge allows us to grow and adapt in our own complex adaptive system—life as we know it.

APPENDIX A – AN INTRODUCTION TO CHAOS, COMPLEX ADAPTIVE SYSTEMS THEORY, AND ARTIFICIAL LIFE

A.1. Chaos Theory

A.1.1. The History and Development of Chaos Theory

As we approach the end of the millenium, we tend to reminisce about the great things that were accomplished in the past century. In the sciences, we are now leaps and bounds ahead of where we were only a few generations ago. Looking back on the past 100 years, we can observe that the human race has undergone three main revolutions in the sciences (Gleick, 1987:6). The first revolution occurred when a brilliant German scientist discovered that how we view all things in the universe is relative to the observer's frame of reference. Einstein's theory of relativity changed how we viewed the macrocosm. Next came the revolutionary ideas of quantum mechanics, which changed the way we viewed the smallest entities in nature, the very building blocks upon which everything else is created. And the third revolution, which many argue bridges the gap between the previous two revolutions, is the emergence of the new sciences and chaos theory.

Chaos theory completely changed the way in which humanity viewed nature. It allowed us to see order where before we saw nothing but randomness, and it led us to see random-like behavior that was actually very ordered. In a paradox of logic, chaos meant that it was possible to derive order out of randomness, predictability out of stochastic processes. It changed the way everything was viewed.

The history of chaos theory only goes back a few decades. James Gleick, in his renowned text Chaos (see Gleick, 1987), presents an outstanding story about the development of this fascinating science. According to him, it all started in the winter of 1961, a meteorologist named Edward Lorenz became the first person to identify, or discover, what came to be known as chaos. Using his Royal McBee weather model, Lorenz found that the model's output varied considerably by only changing ever so slightly the data he used as inputs into the model—the changes were on digits in the thousandths and ten-thousandths places of his inputs. Dubbed 'sensitivity to initial conditions' (SIC for short), this became one of the calling cards of chaotic behavior. Lorenz went on to experiment with a water-wheel which was in essence a mechanical analogue of the rotating circle of convection found in any mixing process. Using some basic differential equations, Lorenz was able to model the process on a computer and create a three-dimensional graph of the system's behavior. What was born was the Lorenz attractor, a two dimensional representation of which is presented by Susan Durham in her paper "Chaos Theory for the Practical Military Mind" (see Figure 9: Lorenz Attractor). The resulting graph displayed an image that stayed within certain

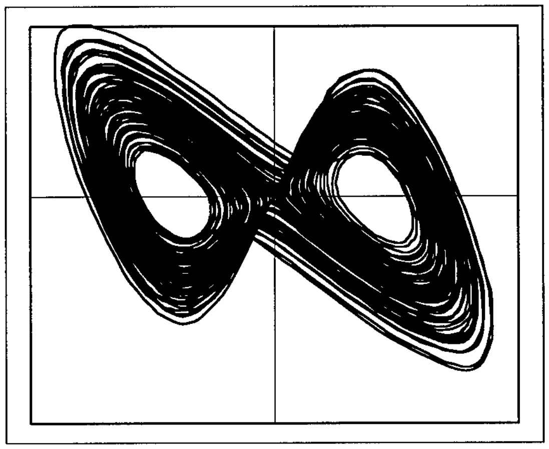


Figure 9: Lorenz Attractor

bounds, never running off the page, but also never repeating itself. It traced a strange, but distinctive shape on the computer screen. To the observer, the Lorenz shape "signaled pure disorder, since no point or pattern of points ever recurred. Yet it also signaled a new kind of order" (Gleick, 1987:30). The theory of chaos was emerging.

In the 60's, the theory of chaos really didn't flourish as everyone would have hoped. The scientific community was too caught up in deterministic pursuits to waste any precious time or money on studying things that were random. Some brave souls, however, did venture into this new arena, and it is because of them that chaos theory is what it is today. Stephen Smale at the University of California at Berkley began experimenting with nonlinear (chaotic) oscillators. Using the mathematical concept of topology, he discovered a system that was locally unpredictable but globally stable. This apparent contradiction of terms would take him years to figure out, but at least he was heading in the right direction.

By the 70's, chaosologists, as they were referred to, were beginning to come out of the closet. Robert May at the Institute for Advanced Study in Princeton began creating bifurcation diagrams in 1971 using logistic equations. He plotted the level of one parameter (along the horizontal axis) against the population of the system (along the vertical axis). His bifurcation diagrams showed how changes in one parameter would change the ultimate behavior of the system, and how after a certain point, what he called the 'point of accumulation', periodicity in the system gave way to chaos where fluctuations occurred that would suddenly settle into a periodic rhythm and then disappear again. This startling discovery meant that chaos could emerge and disappear again in the same system over time. This also meant that perhaps chaos could be controlled in some manner. A startling notion in its own right.

That same year, David Ruelle at the Institut des Hautes Études Scientifiques used phase space plots to capture all the dynamic information of a system in the trajectories of phase space lines. His diagrams created 'strange attractors', areas where the trajectories created by the phase space lines would be found. These strange attractors contained the important qualities of being stable, low-dimensional, and nonperiodic. In essence, the strange attractor was the graphical depiction of chaos. The problem now was to find such behavior in the real world and not just in the theoretical world.

Later in the 1970's, a scientist at IBM helped make the first steps towards seeing chaos in nature. Benoit Mandelbrot was unhappy with the integer based dimensional system that humans had adopted. Nature did not work in strict integers, and he was convinced there was something else out there. After countless experiments and much research, he came up with the concept of fractals, or fractional dimensions. His new geometry "mirrored a universe that was rough, not rounded, scabrous, not smooth" (Gleick, 1987:94). He had pointed out how the pits and tangles that we assume to be negligible are actually more than distorting blemishes, they are the keys to the essence of the entity being viewed.

During this time the universality of chaos theory was also being uncovered. Working at the Los Alamos National Laboratory, Mitchell Feigenbaum began examining the ratio of convergence in the scaling patterns of chaotic systems. To his surprise, no matter what system he looked at, he discovered that their convergence ratio was a constant, which he dubbed the Feigenbaum constant, equal to 4.6692. Albert Libchaber and Jean Maurer at the École Normale Supérieure confirmed the existence of this constant in an experiment concerning the changing convection patterns in a small helium box.

With Feigenbaum's discovery of a constant aspect shared by all chaotic systems and Libchaber's experimental confirmation of this result, chaos theory really began to take off. At Santa Cruz University in 1978, the dynamical systems group was formed as Robert Shaw, Doyne Farmer, Norman Packard, and James Crutchfield began experimenting with Lyapunov exponents as a means of describing strange attractors. Discovering that all strange attractors had at least one positive Lyapunov exponent, this group of men began seeing chaos in everything around them. Shaw went on to conduct experiments that involved simple systems like a dripping water faucet. Instead of modeling the complex physics of the system, he used only the data it produced and embedded that data into a phase space plot to discover the system's strange attractor. This breakthrough made chaotic concepts available to everyone, and the 1980's saw the emergence of chaos theory in all kinds of arenas, from physiology and medicine to the social sciences and economics. Shaw's results proved that an understanding of the system was secondary to a means of producing data from the system. The system could be regarded as a black box so long as it produced usable data which could then be embedded into a phase space plot of enough dimensions. The ability to examine complex systems with simple tools appealed to hundreds of scientists throughout the world. Chaos theory finally got the recognition it deserved, and by the 1980's nearly everyone was trying to find a way to apply it to their research.

A.1.2. The Tenets of Chaos Theory

The concept of chaos still eludes many people. Everyone even remotely familiar with systems behavior knows that some systems act in a very deterministic, periodic way and others act in a completely stochastic, random way. But, the line between periodicity and randomness is somewhat fuzzy. This fuzzy area is the arena of chaos. Chaotic systems lie between periodic systems and random systems. They involve aspects of both, but belong to neither. Chaotic systems exhibit behavior that never repeats itself, an aspect of randomness, but patterns do evolve around attractors and long term trends are predictable, an aspect of periodic behavior. So just what makes up a chaotic system?

Many different 'experts' have their own claims about what make a system chaotic, but throughout the literature on chaos, the following tenets of chaos seem to hold. All chaotic systems are:

- deterministic – they are not random;
- nonlinear – the whole is greater than the sum of the parts;
- sensitive to initial conditions – known as SIC;
- bounded – strange attractors create what are known as basins of attraction which allow for the prediction of long-term trends;
- aperiodic – although long-term trends can be identified, short-term predictions are all but impossible;
- controllable – a system can be driven in and out of chaos.

These conditions represent the necessary and sufficient conditions for a system to be considered chaotic. Therefore, every chaotic system possesses these traits and every systems that possesses these traits is chaotic.

Added to the above listed characteristics of chaotic systems, there are also a number traits that chaotic systems have in common. In his book Chaos Theory, Glenn James points out the following observable features common to all chaotic systems (James, 1996:38):

- transient and limit dynamics – the system is constantly changing, but these changes occur in a limited arena (i.e. the attractor's basin of attraction);
- parameters – the system has control knobs that will alter the system's dynamics;
- definite transitions to and from chaotic behavior – the system exhibits the ability to jump in and out of chaos;
- attractors – the system's attractors often possess fractal dimensions.

What at first was considered random noise in the data has now been determined to be the existence of chaotic behavior in the system. This has made for revelations in countless arenas of the scientific community. As Gottfried Mayer-Kress points out, "the primary feature distinguishing chaotic from random behavior is the presence of an attractor that outlines the dynamics towards which a system will evolve" (quoted in James, 1996:39). The existence of such an attractor lends hope to the notion that the dynamics of a chaotic system are in fact repeatable. Despite the fact that chaotic systems are very sensitive to initial conditions and that neighboring trajectories actually repel each other, the trajectories of a chaotic system confine themselves to some limited region of phase space. This bounded region has minimum and maximum parameter values beyond which the trajectories will not wander unless perturbed by some outside force. Strange attractors also possess the mixing property which states that trajectories in phase space will repeatedly pass near every point on the attractor. This means that strange attractors are efficient mixers.

The strange attractor for a chaotic system forms a basin of attraction as all the transient states in the system evolve toward a single attractor. The basin then comes to represent all the possible initial states that ultimately exhibit the same limit dynamics on the attractor. As stated above, because of the system's sensitivity to initial conditions, the precise state of the system at any given time cannot be predicted. But, because the attractor draws trajectories towards itself, we do know what the trends in the dynamics of the system will be. Thus, an attractor shows not only distributions of system states, but it also indicates directional information about how the system tends to change from its current state. As a result, when an attractor is constructed, an image of the system's global dynamics (without appealing to any model) is created. This reconstruction allows us to predict (very) short-term trajectories and overall long-term trends. We can also perform pattern recognition exercises and carry out sensitivity analysis due to the existence of a strange attractor.

The presence of a strange attractor means that chaotic flow in the system settles into a boundary region. The chaotic flow in the system generates time intervals with no periodicity and no apparent pattern. Thanks to the strange attractor, though, the chaotic return map does not simply fill all the available space with a random smear of points. The strange attractor ensures that there is some rough (often fractal) boundary confining all the chaotic points, even if the points appear to fill the region in an erratic, unpredictable way.

All chaotic systems have a key control parameter that allows the regulation of the amount of energy in the system. Chaotic behavior appears when the system has so much energy that there is insufficient time for it to relax and recover before the next event occurs. Since the transition to more complicated behavior can occur at predictable parameter values (aka knob settings), we can expect specific behaviors in a system that displays periodic behavior. We can also expect to see higher periods and eventually chaotic behavior in the system as additional energy is added. In some cases we may even be able to forecast the parameter values that permit these transitions from periodicity to chaos. This leads to the notion that chaos can be controlled. Just as perturbing a stable system can induce chaos, perturbing a chaotic system can stabilize the system's behavior. This alone makes the study of chaos a worth-while cause, since chaotic behavior can be both advantageous and disadvantageous, depending on the circumstances surrounding the system in question.

Lastly, the idea of feedback and nonlinearity lends chaos theory to a plethora of natural systems. Nature, contrary to popular belief, is not strictly linear. In our lack of diligence, humans often force our linear models on nature's nonlinear reality. As such, chaos has been hidden for centuries in what was considered experimental error or noise. Feedback in a system has only recently been taken advantage of, and thanks to chaos theory, there is now a great mechanism for examining just that principle. Feedback means that a system has the ability to correct itself, either in a positive way or a negative way. This idea leads to learning on the system's part and allows us to examine the possibilities of intelligent systems. The great potential dynamics that can be generated by imposing feedback on a system will lend a lot to the study of many natural occurrences and open many new doors to scientific understanding.

One of the greatest things about chaos theory is that it can be applied to a system without the need to know any of the underlying physical relationships of the internal make-up of the system. As such, even very complicated systems can be examined using chaos theory, so long as the system produces a reliable, steady stream of data. The uses of chaos theory are unlimited, and as James Gleick puts it so eloquently, "now that science is looking, chaos seems to be everywhere" (Gleick, 1987:5).

A.1.3. The Uses and Applications of Chaos Theory

Looking at what spawned the whole notion of chaos, namely the investigation of turbulence, it is easy to see that any system involving turbulence involves chaos to some degree. Just a fraction of the systems that involve turbulent behavior include those pertaining to:

- air flows over wings (Durham, 1997:34);
- water flows along the bow of a ship (Durham, 1997:34);
- atmospheric turbulence and its effect on electronic imaging (Durham, 1997:35);
- turbulence encountered in the mixing of fluids (Durham, 1997:35);
- lasers themselves have nonlinear fluctuations that are turbulent (Durham, 1997:35).

Any system in which vibrations are an important aspect of the effective use of the entity are havens for chaos. Some systems that might involve chaotically behaving vibrations include:

- electronic or mechanical vibrations in computer equipment (Durham, 1997:36);

- engine vibrations (James, 1996:73);
- helicopter vibrations (James, 1996:73).

Outside the realm of turbulence or vibrating systems, there are still countless examples of systems in which an understanding of chaos theory can be priceless. Just a few of the systems that exhibit chaotic behavior include:

- systems in the human body – especially the beating of the heart (Durham, 1997:36);
- transient phenomena such as waves and their effects on the stability of ships (Durham, 1997:36);
- systems made of electronic circuits – feedback in audio and electrical systems can drive the system in and out of chaos (Durham, 1997:36);
- systems in which efficient mixing is required – as stated above, strange attractors have the property of being efficient mixers (James, 1996:73-75);
- flickering lasers or any other pulsating electro-magnetic energy source (James, 1996:73-75).

Taking a closer look at the fractal nature of chaotic systems, the use of fractal technology can be exploited in such arenas as image compression and pattern recognition (James, 1996:84-86). David Nicholls, in his article "What Does Chaos Theory Mean for Warfare", points out that chaos theory can be used to define the minimum number of variables required to create a model of warfare (Nicholls, 1994:53). He claims that the fractal dimension of a system is roughly equivalent to the number of variables needed to characterize the system. Therefore, the fractal nature of chaotic systems may allow relatively small and simple models to accurately simulate complex systems such as the modern battlefield.

Chaos theory's tenet of sensitivity to initial conditions, Nicholls points out, could also help modelers determine the predictive confidence with which they may use their models. If small changes in the initial conditions produce small variations in the system, then predictions can be counted on more, but if the same small changes in initial conditions produce large variations in the system, then predictions should be suspect.

Chaos theory's applications to the military sciences are also being exploited in many other arenas. Nicholls claims that his process of altering initial conditions will help identify centers of gravity in the enemy's system. He also describes how chaos theory can be used to identify sources of nonlinearity in warfare, among which might include:

- feedback loops;
- the psychology associated with interpreting enemy actions;
- the processes that are by themselves inherently nonlinear;
- Clausewitzian fog and friction;
- the process of decision making itself.

The notion of chaos theory explaining uncertainty in warfare actually goes all the way back to the Clausewitzian idea of friction. In his article "Clausewitz, Nonlinearity, and the Unpredictability of War", Alan Beyerchen shows how Clausewitz unintentionally used the concepts of chaos theory to explain the unpredictability inherent in warfare (Beyerchen, 1992/1993:90). Beyerchen claims that Clausewitz just didn't possess the vocabulary to explain his ideas in the terms of chaos theory, and therefore called his uncertainty the fog and friction of war. Clausewitz's theory of the trinity of war is also based on the idea of a chaotic system. The trinity of war, representing the violence of the masses, the chance opportunities made available to the military to defeat the enemy, and the reasoning abilities of the government to avoid our evolution into total war, is always in flux between three opposing poles of attraction. This state of flux exhibits characteristically chaotic behavior. Added to this, the nonlinearity of war itself makes it a very chaotic system.

Wayne Hughes, in his article "Uncertainty in Combat" shows how the fruits of chaos theory in combat are that "macro patterns [are] seen amidst micro unpredictability" (Hughes, 1994:51). He shows that for a commanding general:

> A major part of skill and expertise is recognizing and avoiding situations dominated by uncertainty or chance when your knowledge is superior, and creating opportunities for uncertainty for the enemy when your knowledge is inferior. (Hughes, 1994:50)

Understanding that uncertainty cannot be eliminated from combat, Hughes espouses a strategy for learning how to deal with it and how to exploit it. He claims that a commander's best defense against uncertainty is to pursue a "recipe of breadth, not depth, of analysis" (Hughes, 1994:52). In doing so, a commander learns to recognize chaotic behavior in both his systems and the enemy's systems, and through experience in dealing with it in the past, he knows what to do to exploit it for his own use.

Piggy-backing on this idea, John McDonald, in his article "Exploiting Battlespace Transparency – Operating Inside an Opponent's Decision Cycle" claims that for a commander to be victorious on the battlefield he must endeavor to get into his enemy's observe-orient-decide-act (OODA) cycle and induce chaos into it (McDonald, 1997). McDonald shows that the way to do this is to make your decisions faster and better than your enemy's. The commander must also learn to avoid uncertainty in his decision cycle and employ techniques to induce it in his enemy's, therefore causing his enemy's systems to act chaotically.

Lastly, James presents a multitude of examples through which the exploitation of chaos theory can help in the military setting (James, 1996:79-84). Among these are:

- induce chaotic behavior in the enemy's stable systems;
- reduce and/or eliminate chaos in our own chaotic behaving systems;
- exploit chaos theory in our decision making tools;
- use chaos theory as a means for recognizing patterns;
- employ chaos in feedback systems – 'ping' the enemy's system and see how it responds.

Durham, citing Jim Lesurf's Chaos on the Circuit Board, also includes one interesting use of chaos theory for the military specialist—encoding data to make it sound like 'random' noise (Durham, 1997:36).

A.2. Complex Adaptive Systems Theory

A.2.1. The History and Development of Complex Adaptive Systems Theory

As the use of chaos theory grew to include many other branches of science, some practitioners began to see extensions of its use. If complex systems could be understood by just looking at the data produced, why not build simple models to simulate these complex systems. Nature was full of these types of simple agents that interacted in very complex manners. Birds flocking, flies swarming, or schools of fish swimming in the ocean are just a few such examples. As it turned out, making these models was not as complicated as one would expect and soon complex systems theory would lead to an even better understanding of the natural world.

In reality, complex adaptive systems theory predated the development of chaos theory. Famed scientist John Von Neumann played with agent-based simulations long before Lorenz even thought about examining chaos. However, the advent of chaos theory, coupled with its wide-spread acceptance as a viable science, has spurned new interest in complex adaptive systems theory. In essence, the discovery of chaos theory has led to a greater understanding of complex adaptive systems theory. Chaos theory presented researchers with insights into complex adaptive systems theory that they never had, or could have had before.

Complex adaptive systems seemed to bring order and chaos into balance. It was at this edge of chaos, as M. Mitchell Waldrop points out, that the science of complexity emerged:

> In the past two decades, chaos theory has shaken science to its foundations with the realization that very simple dynamical rules can give rise to extraordinarily intricate behavior; witness the endlessly detailed beauty of fractals, or the foaming turbulence of a river. And yet chaos by itself doesn't explain the structure, the coherence, the self-organizing cohesiveness of complex systems. Instead, all these complex systems have somehow acquired the ability to bring order and chaos into a special kind of balance. This balance point—often called the edge of chaos—is where the components of a system never quite lock into place, and yet never quite dissolve into turbulence, either. The edge of chaos is where life has enough stability to sustain itself and enough creativity to deserve the name of life. The edge of chaos is where new ideas and innovative genotypes are forever nibbling away at the edges of the status quo, and where even the most entrenched old guard will eventually be overthrown. (Waldrop, 1992:11-12)

Stuart Kauffman adds that "the fate of all complex adaptive systems in the biosphere—from single cells to economies—is to evolve to a natural state between order and chaos, a grand compromise between structure and surprise" (Kauffman, 1995:15).

Taking the idea of strange attractors from chaos theory, it came as no surprise to find that complex adaptive systems tend to have strange attractors. Looking again at the reality of nature, systems with strange attractors are the rule, not the exception. It's just that science has always called their behavior random when in fact there was order hidden underneath. Exploiting the ideas of chaos and complexity together finally allowed us to uncover some degree of that hidden order.

One of the greatest advantages about using complex adaptive systems theory to model behavior was that it was based on a bottom-up approach to modeling, as opposed to the top-down approach so common in modeling over the past few decades. But, as Waldrop points out, "after three hundred years of dissecting everything into molecules and atoms and nuclei and quarks, they finally seemed to be turning that process inside out. Instead of looking for the simplest pieces possible, they were starting to look at how those pieces go together into complex wholes" (Waldrop, 1992:16). As John Casti states, "instead of dictating the rules of thought, a Bottom Up researcher sees whatever rules of thought the machine embodies as emerging form the reconfiguration of the pattern connecting the many processors as the machine learns to survive in its environment" (Casti, 1994:159). This then allows the complex adaptive system to learn and adapt to its environment. Based on only a small set of rules, the system is able to cope with an ever changing environment and learn how to deal with adversity—something that no other model to date had been able to do.

The science of complexity was changing the way everyone was looking at nature. It was a complete paradigm shift in the way models were built. It was a revolution in the sciences, one that began according to Brian Arthur, the first time someone said, "Hey, I can start with this amazingly simple system, and look—it gives rise to these immensely complicated and unpredictable consequences" (quoted in Waldrop, 1992:329)

The center for most, if not all of the research that has gone into the sciences of complexity is arguably located in Santa Fe New Mexico. Established in May of 1984, the Santa Fe Institute was founded as a "small, private organization set up by the physicist Murray Gell-Mann and others to study aspects of complex systems, by which they meant everything from condensed-matter physics to society as a whole—anything with lots of strongly interacting parts" (Waldrop, 1992:53). A think tank set up initially by grants from private industries, primarily from the financial sectors of the community, the Santa Fe Institute (SFI) was tasked originally with creating better models of the economy. The goal was to make money, but soon its founders and members saw a much bigger catch. They hoped and dreamed of creating a whole new science, one that would draw all the other sciences together. As Waldrop so eloquently puts it:

> They [members of the Santa Fe Institute] all share the vision of an underlying unity, a common theoretical framework for complexity that would illuminate nature and humankind alike. They believe that they have in hand the mathematical tools to create such a framework, drawing from the past twenty years of intellectual ferment in such fields as neural networks, ecology, artificial intelligence, and chaos theory. They believe that their application of these ideas is allowing them to understand the spontaneous, self-organizing dynamics of the world in a way that no one ever has before—with the potential for immense impact on the conduct of economics, business, and even politics. They believe that they are forging the first rigorous

> alternative to the kind of linear, reductionist thinking that has dominated science since the time of Newton—and that has now gone about as far as it can go in addressing the problems of our modern world. They believe they are creating, in the words of Santa Fe Institute founder George Cowan, "the sciences of the twenty-first century" (Waldrop, 1992:12-13).

Probably the reason for the length of time it took for the science of complexity to really take off can be attributed to the immense amount of computational power needed to effectively analyze complex adaptive systems. The power of the computer, along with the relative inexpensiveness of said computing power especially in the past 15 years, definitely helped open the doors to studying complexity. In the past, solving mathematical equations by hand limited scientists to three or four linearly related variables. But computers do not differentiate much between linear and nonlinear systems. Their faster processors allow for millions, if not billions, of calculations per minute. Added to this, computer graphics allow for the dynamic display of data in colorful, visually appealing graphs. Finally, the easy access to personal computers allowed everyone to use the power of computers anywhere they wanted to. The recent proliferation of super-computers, especially at universities and colleges throughout the country, also allowed for the investigation of really big problems.

What makes computers great for studying complex adaptive systems is not just their mathematical prowess, but also their record-keeping abilities. They are world-class bookkeeping machines, and, as Waldrop points out, "properly programmed, computers could become entire, self-contained worlds, which scientists could explore in ways that vastly enriched their understanding of the real world" (Waldrop, 1992:63). The whole notion of using complex adaptive systems theory to create simulations allowed scientists to start tackling very complex problems they would not have dared even think about in the past.

The idea of nonlinear dynamics introduced by chaos theory also played very well into the science of complexity. With computers able to easily tackle complex mathematics, the need for strict linearity assumptions dwindled away quickly. Since computers could handle nonlinear problems with ease, the door was opened to discarding all the linearly imposed assumptions that were required in the hand calculation days. Nonlinear dynamics were born, and with it the disturbing notion that the whole could really be greater than the sum of the parts became a reality. Chaos theory introduced this synergistic idea, but complexity exploited it. Complex behaviors were arising when very simple systems followed nonlinear rules. The complexity seemed to emerge from simplicity.

In the earliest days of the Santa Fe Institute, one of the first proponents of the use of complex adaptive systems theory was Brian Arthur. An economics professor from Berkley, Arthur had a big problem with the theories espoused by classical economists who "seemed to see human society as a perfectly oiled machine governed by the Invisible Hand of Adam Smith" (Waldrop, 1992:23). While classical economists believed in equilibrium and the idea that tiny (and even not-so-tiny) imbalances die away, Arthur believed that the economy was a living system—like an organism. To him, the economy was "an entire sprawling set of self-consistent patterns that formed and evolved and changed in response to the outside world" (Waldrop, 1992:31). Arthur saw an economy where tiny initial differences produced enormously different results. The simple dynamics of supply and demand led to astonishingly complex behavior patterns.

Following the ideas of complex adaptive systems, Arthur saw the economy as a collection of constantly interacting agents. Again, going against the grain of the classical economists, Arthur pictured his economic agents as smart, but not perfect. The classical economic idea of bounded rationality—that all agents act perfectly rationally on perfectly accurate information—seemed ludicrous to Arthur. Additionally, the notion of bounded rationality was completely unrealistic and in-human, hence classical economic theory did nothing to accurately explain, let alone predict the economy. Using the ideas of complex adaptive systems theory, Arthur built economic agents that were smart, but not perfect. He allowed his agents to learn from experience garnered from their interactions with each other and their environment. He sought to recreate the economy and allow free market principles, like bartering and the use of money, to emerge from a group of fairly simple agents. The complex adaptive system economic agents did not have to be perfect. They could dynamically change their behavior as they learned from their experiences. They would be allowed to act more like individuals in a real economy would act. In short, an economic model based on complex adaptive systems theory might finally help explain the behavior of the economy as it really was, not as the classical economists thought it should be.

The adaptive agents employed by Arthur operated in an inductive mode. They were forced to reason an internal model from fragmentary data. Instead of deducing answers by ripping apart the system into its smaller parts, the agents induced answers by combining the parts of the puzzle they had and then taking a best guess at what to do with the information they had. The entire notion of induction fascinated Arthur:

> Here you could think about doing economics where the problem facing the economic agent was not even well defined, where the environment is not well defined, where the environment might be changing, where the changes were totally unknown. And, of course, you just had to think for about a tenth of a second to realize, that's what life is all about. People routinely make decisions in contexts that are not well defined, without even realizing it. You muddle through, you adapt ideas, you copy, you try what worked in the past, you try out things. (Waldrop, 1992:254)

This drastic shift also helped make the complex adaptive systems approach to modeling the economy, or any complex system, more realistic. Additionally, it allowed for the formation of models built using imperfect information. Perfect information was not necessary for induction to occur, but it was a requirement of deduction.

The Santa Fe approach to economics became the first of many paradigm shifts in the sciences that would result from applying the ideas of complex adaptive systems theory to practice. Unlike the static and unrealistic model of the economy proposed by the classical economists, the Santa Fe approach emphasized increasing returns (the whole is greater than the sum of the parts), bounded rationality (agents weren't perfect), and the dynamics of evolution and learning. The models created were psychologically realistic and treated the economy as something organic, adaptive, surprising, and alive. They talked about the economy as a dynamic, ever-changing system poised at the edge of chaos. Gone were the days of static equilibrium, and here to stay were the days of reality and complexity.

Like any revolutionary idea, Arthur's ideas came under some heavy criticism. Among the strongest was the fact that because of the chaotic nature of complex adaptive systems, the notion of prediction was all but impossible. Arthur, however, cited the proponents of chaos theory and their answer to this dilemma by pointing out that prediction wasn't the goal, explanation was: "Prediction isn't the essence of science. The essence is comprehension and explanation. And that's precisely what Santa Fe could hope to do with economics and other social sciences" (Waldrop, 1992:255). So even though Arthur's ideas made prediction essentially impossible, they did explain the nature of the economy and led to a model that was realistic and based on the true system. Sacrificing prediction allowed him to concentrate on explanation. In reality, the sacrifice wasn't really that big since classical economic models are not good prediction tools and poorly explain how the economy really works. In the end, the complex adaptive systems theory approach to modeling the economy was the better approach.

Once the scientific community understood that the use of complex adaptive systems theory led to models that contained lots of explanatory power and not many predictive capabilities, they began to see the potential for its use in a lot of different arenas. Arthur's work in the field of economics was only one, albeit the first, such venture into the realm of complex adaptive systems theory thought. The door was now open to explore the use of complex adaptive systems in a variety of different arenas.

At about the same time that Brian Arthur was working on economic models that employed complex adaptive systems theory, John Holland was refining the definition of what made up the characteristics of a generic complex adaptive system. A professor of Electrical Engineering and Computer Science at the University of Michigan, Holland had made a name for himself as the father of genetic algorithm. For years he had sought a means to put his genetic algorithms to work, and with the advent of complex adaptive systems theory, he saw his opening. Heavily influenced by biology and the works of

Darwin, Holland saw complex adaptive systems theory as a great way to explain the theory of evolution. But first he would have to solidify some of the general characteristics shared by all complex adaptive systems.

To Holland, all complex adaptive systems should have a set of shared characteristics, rules of play. He decided to set out and work on defining these rules so as to bring some consistency to the science of complexity. First and foremost, a complex adaptive system had to be composed of a network of autonomous agents that acted in parallel. These agents acted in a system whose control was dispersed among the actual agents; there were no God-like oracles running the system. This allowed for system-wide behavior to emerge from the interactions agents had with each other and with their environment.

Although Holland's agents were autonomous and self-governing, he also realized that different levels of organization were required for the system to be able to evolve to a higher state. He claimed that "a complex adaptive system has many levels of organization, with agents at any one level serving as the building blocks for agents at a higher level" (Waldrop, 1992:145). These building blocks were constantly being revised and rearranged based on the experiences the agents gained from their interactions with each other and their environment. In a manner of speaking, the agents organized themselves based on the knowledge they acquired by living in the system.

Lastly, Holland's agents used the notion of anticipation of the future as a means to drive their immediate behaviors. He developed the idea of internal models as a basis for making predictions about the future. These internal models are in a state of constant and active evolution in which learning takes place. Employing cycles of prediction and feedback, the agents learn from their environment. They make anticipations based on their internal models and learn, or adapt, to the feedback they receive from their environment. As a result, they develop rules of conduct that help them interact with each other and their environment. These rules, Holland postulated, are organized into hierarchies which are then further organized into higher levels of a hierarchy and so on. This was a prime place for his genetic algorithms to come in to play and for evolution to begin.

Holland's ideas meant that the agents of a complex adaptive system could fill their own niches. He claimed that niches were present in all types of complex adaptive systems. Furthermore, the very act of filling one niche opens up a number of other niches. This all taken together helped him postulate the idea that a complex adaptive system never reaches equilibrium:

> [because of the niche idea above, it] means that it's essentially meaningless to talk about a complex adaptive system being in equilibrium: the system can never get there. It is always unfolding, always in transition. In fact, if the system ever does reach equilibrium, it isn't just stable. It's dead... There's no point in imagining that the agents in the system can ever "optimize" their fitness, or their utility, or whatever. The space of possibilities is too vast; they have no practical way of finding the optimum. The most they can ever do is to change and improve themselves relative to what the other agents are doing. In short, complex adaptive systems are characterized by perpetual novelty. Multiple agents, building blocks, internal models, perpetual novelty—taking all this together, said Holland, it's no wonder that complex adaptive systems were so hard to analyze with standard mathematics. Most of the conventional techniques like calculus or linear analysis are very well suited to describe unchanging particles moving in a fixed environment. But to really get a deep understanding of the economy, or complex adaptive systems in general, what you need are mathematics and computer simulation

> techniques that emphasize internal models, the emergence of new building blocks, and the rich web of interactions between multiple agents. (Waldrop, 1992:147)

In the end, however,

> what captivated him [Holland] wasn't that science allowed you to reduce everything in the universe to a few simple laws. It was just the opposite: that science showed you how a few simple laws could produce the enormously rich behavior of the world. (Waldrop, 1992:153)

And this fact is what made the science of complexity so appealing to so many others. Aside from the few characteristics described above by Holland, anyone could use complex adaptive systems theory to model a dynamic system. And that is just what started to happen. All over the world, scientists began using the ideas found in complex adaptive systems theory to form their own complex adaptive systems based on simple rules. And to everyone's astonishment, these complex adaptive systems helped start explaining things accurately that were being taken for granted in the natural world.

It did not take long for the theory of complex adaptive systems to enter into the mainstream. Once the tenets of the theory were laid out, people from all over society, not just from the sciences, experimented with it. One of these outsiders was Craig Reynolds. A computer programmer and animation specialist for the Symbolics Corporation of Los Angeles California, Reynolds saw the theory of complex adaptive systems as a means to add life to his animation. Trying for years to re-create the flocking behavior of birds in flight, he saw complex adaptive systems theory as the solution he had been looking for. Dubbing his computer generated agents boids, he made three simple rules by which each and every boid was required to live by:

1. separation – maintain a minimum distance from other objects in the

environment, including other boids;

2. alignment – try to match velocities with boids in its neighborhood;
3. cohesion – try to move toward the perceived center of mass of the boids in its neighborhood.

After defining these simple rules for each boid, the system was then released to act on its own. What happened next was nothing short of amazing. These simple rules generated the flocking behavior exhibited daily by birds in the natural world:

> What was striking about these rules was that none of them said, "Form a flock." Quite the opposite: the rules were entirely local, referring only to what an individual boid could see and do in its own vicinity. If a flock was going to form at all, it would have to do so from the bottom up, as an emergent phenomenon. And yet flocks did form, every time. Reynolds could start his simulation with boids scattered around the computer screen completely at random, and they would spontaneously collect themselves into a flock that could fly around obstacles in a very fluid and natural manner. (Waldrop, 1992:241-242)

Reynolds simple example was the coup the complex adaptive systems community had been waiting for. Here was an incredibly simple system that created very complex behavior, behavior that to date had not really been successfully recreated in a computer model. Reynolds had done in a few lines of code what computer scientists could not do with pages of complicated algorithms. Viewing Reynolds boids, even the casual observer could recognize the emergent behavior it displayed. And the greatest thing about his program was that:

> Instead of writing global, top-down specifications for how the flock should behave, or telling his creatures to follow the lead of one Boss Boid, Reynolds had used only the three simple rules of local, boid-to-boid interaction. And it was precisely that locality that allowed his flock to adapt to changing conditions so organically. The rules always

> tended to pull the boids together, in somewhat the same way that Adam Smith's Invisible Hand tends to pull supply into balance with demand. But just as in the economy, the tendency to converge was only a tendency, the result of each boid reacting to what the other boids were doing in its immediate neighborhood. So when a flock encountered an obstacle such as a pillar, it had no trouble splitting apart and flowing to either side as each boid did its own thing. Try doing that with a single set of top-level rules, said Langton. The system would be impossibly cumbersome and complicated, with the rules telling each boid precisely what to do in every conceivable situation. In fact, he had seen simulations like that; they usually ended up looking jerky and unnatural, more like an animated cartoon than like animated life. And besides, he said, since it's effectively impossible to cover every conceivable situation, top-down systems are forever running into combinations of events they don't know how to handle. They tend to be touchy and fragile, and they all too often grind to a halt in a dither of indecision. (Waldrop, 1992:279)

To Brian Arthur this meant that "if Reynolds could produce startlingly realistic flocking behavior with just three simple rules, then it was at least conceivable that a computer full of well-designed adaptive agents might produce startlingly realistic economic behavior" (Waldrop, 1992:243). Reynolds success with boids seemed to finally make the notion of complex adaptive systems available for commercial use, but the best was yet to come.

In 1989, a small computer programming firm from Silicon Valley began experimenting with complex adaptive systems in their games. The Maxis Company of Orinda California created the program SimCity that year, and it became an instant success. Based on the idea of building a city from the ground up, SimCity employed complex adaptive systems theory to help generate the city. The simulation involved the player taking on the role of a mayor who tried to build his city to prosperity in the midst of crime, pollution, traffic congestion, and tax revolts. Considered just a game, it ironically drew the attention of real city planners who actually started using it in their city designs. To Holland, it was proof that non-scientists could use a model to test out ideas before actually implementing the ideas in the real world. A computer model based on complex adaptive systems theory seemed to be the best way to go, especially since it required defining only a (relatively) few simple rules and its output was incredibly realistic.

As the Santa Fe Institute entered the 1990's it seemed as if the notion of complex adaptive systems theory had finally caught on. But one more piece of the puzzle would be added before everyone was happy with the result, and that piece would be contributed by Chris Langton, a doctoral student at the University of Michigan working with Holland. Langton, a computer scientist, decided to take the notion of artificial intelligence one step further than it had been taken in the past. Instead of modeling just the brain and how it 'created' intelligence, he decided to pursue the notion of artificial life and model the entire organism. Langton's vehicle of choice for this great endeavor was complex adaptive systems theory.

Langton felt comfortable considering living systems machines, but to him they were very different from what most people considered machines:

> Living systems are machines, all right, but machines with a very different kind of organization from the ones we're used to. Instead of being designed from the top down, the way a human engineer would do it, living systems always seem to emerge from the bottom up, from a population of much simpler systems. (Waldrop, 1992:278)

This whole idea made it possible for emergence to happen. Complex behavior doesn't need to come from complex roots, it can emerge from strikingly simple components. To

Langton, life may indeed have been a kind of biochemical machine. But to animate such a machine "is not to bring life to a machine; rather, it is to organize a population of machines in such a way that their interacting dynamics are 'alive'" (Waldrop, 1992:280). The complexity and the 'being alive' would therefore be a property of the organization of entities, not the entities themselves. This all fit perfectly into the whole realm of complex adaptive systems theory, giving Langton the means for creating artificial life.

Langton, following in the footsteps of his mentor John Holland, saw a great need to standardize the whole notion of artificial life. As such, he established the criteria he felt necessary to achieve lifelike behavior in a simulated environment. He decided, thanks to complex adaptive systems theory, that it would be best to simulate populations of simple units instead of one big complex unit. This would mean that each agent would possess local control over itself and not require a global God-like oracle directing the living process. Decentralized control and the autonomy of the agents would allow behavior to emerge from the bottom-up instead of being specified from the top-down. Therefore life would evolve and adapt to its environment. There would be no real end-result, because the system would keep adapting to its ever changing environment. Therefore the focus would be on ongoing behavior instead of the final result. All these ideas fit well into the reality of life, living systems are comprised of autonomous agents that act and react based on feedback from their interactions with each other and their environment. Life does not reach an equilibrium state and settle down, it is constantly changing and adapting to the changes going on around it. Life is a complex adaptive system, and Langton felt that he might just be able to recreate it (someday) in an elaborate enough simulation.

Preliminary experiments into the creation of artificial life simulations led Langton to another startling discovery. It seemed that all complex adaptive systems built to simulate artificial life were what he called 'on the edge of chaos'. The edge of chaos was that precarious balance between the forces of order and the forces of disorder. It was here that life was possible, and it was here that Langton knew he would find the key to recreating it on a computer. According to Langton, living at the edge of chaos meant that the system being simulated had to consist of a dense web of feedbacks and regulations, while at the same time leaving plenty of room for creativity, change, and the ability to respond to new conditions. The problem was determining how a system got to the edge of chaos, how it kept itself there, and what does it actually do there. Although these questions were still unanswered, he knew the edge of chaos was where he would find his artificial life. For it was at the edge of chaos that the system would act more like a fluid than like a rigid solid. It was there that it would have the ability to adapt and overcome diversity. He also knew that a system would get to the edge of chaos through adaptation, and that artificial life systems would want to stay at the edge of chaos so that they could continue to adapt themselves. The quest now became how to find that edge of chaos.

Having developed the science of complexity and grounded it well within the tenets of complex adaptive systems theory, the Santa Fe Institute accomplished many of its initial goals. Complex adaptive systems were implemented in a wide range of endeavors and the sciences were beginning to see its benefits. The commercial sector also was beginning to draw on complex adaptive systems theory for help in solving problems and creating better methods for evaluating particularly difficult situations. The Santa Fe Institute had done its job in establishing complexity as a science. Its next task became one of application. Scientists from throughout the world sought fellowships there in the hopes of using complex adaptive systems theory and applying it to their own research. The Santa Fe Institute had become the Mecca of complexity, the Jerusalem of the science of complex systems. It was becoming the object of many scientific pilgrimages.

A.2.2. The Tenets of Complex Adaptive Systems Theory

What differentiates complex adaptive systems (CAS) from other modeling systems is that the actual modeling is done at the most basic agent's level as opposed to modeling at the highest overall or global level. It is the interactions of the agents of a complex adaptive system that provide the information to the modeler, not the actual physical make-up of the model's results. Since the interactions of the agents in a complex adaptive system are governed by the anticipations engendered by learning and long-term adaptation, agent-based simulations of complex systems are predicated on the idea that the global behavior of a complex system derives entirely from the low-level interactions among its constituent agents. By simulating an individual constituent of a complex system as an adaptable agent, one can model a real system as an artificial world populated by interacting processes among these so-called agents. This agent-based approach makes modeling very easy and the behavior exhibited by the models very compelling.

The decentralized self-organization portrayed by many such complex adaptive systems is what makes their use so wide-spread. In essence, the modeler is creating artificial life, breathing existence into simple agents by providing them with rules of behavior. The modeler then removes himself from the system and lets the agents do the rest of the work, observing the behavior that emerges as a result of their interactions with each other and their environment.

The emergent behavior of the CAS provides a means for answering 'what-if' type questions and pursuing varying alternatives in environmental factors. Basically, complex adaptive systems theory allows the modeler to play God and create life and then add circumstances that will effect the interactions of the life he has created. The modeler can then view how his creation deals with the circumstances thrown at it.

John Holland is arguably considered the father of the entire complex adaptive systems approach to modeling. In his book Hidden Order, he sets forth the basic properties and mechanisms by which all complex adaptive systems should be built. The four properties, and sub-properties, he describes are:

- aggregation – aggregate simple things into categories and then treat them as equivalent;
- nonlinearity – the whole is more than the sum of the parts;
- flows – flows through a network vary over time as nodes and connections appear and disappear as the agents adapt or fail to adapt to the changing environment resulting in:
    - the multiplier (aka amplifier) effect – creates chain reactions;
    - the recycling effect – preserves resources;
- diversity – each kind of agent fills a niche
    - convergence – if an agent filling a unique niche is removed from the system, some other agent will adapt to fill that niche.

The three mechanisms Holland describes that use these properties are:

- tags - a mechanism for aggregation and boundary formation enabling us to observe and act on properties previously hidden by symmetries;
- internal models – mechanisms for anticipation:
    - tacit models – prescribe a current action under an implicit prediction of some desired future state;

- overt models – used as a basis for explicit, internal explorations of alternatives;

- building blocks – serve to impose regularity on a complex world.

Having defined the above four main properties and three primary mechanisms common to all complex adaptive systems, Holland describes other elements these systems have in common. The first such element is the existence of a performance system, or a uniform way to represent the capabilities of different kinds of agents. Such a performance system includes:

- a set of detectors – to represent the agent's capabilities for extracting information from its environment using detectors that detect (and filter) information from the environment;

- set of IF/THEN rules – to represent the agent's capabilities for processing information;

- set of effectors – to represent the agent's ability to act on its environment by describing the agent's actions on the environment in response to a message.

Once a performance system is established, Holland claims that a means for credit assignment must be created. Credit assignment uses the agent's successes (or failures) to assign credit (or blame) to parts of the performance system. The basic idea behind credit assignment is to assign each rule a strength that over time comes to reflect the rule's usefulness to the system. This allows the system to use a primitive type of utility theory and maximize it's fitness (or life strength) by pursuing those actions that increase its fitness (i.e. give it credit) and avoid those actions that decrease its fitness (i.e. take away credit). In essence, Darwinian survival of the fittest results.

The whole notion of credit assignment leads to the formation of default hierarchies. Obviously, useful general conditions (aka defaults) are relatively easy to find and establish. However, more specific exception rules take progressively longer to find and establish. Therefore, under credit assignment, agents early on depend on over-general default rules that serve better than random actions. As experience accumulates, however, these internal models are modified by adding competing, more specific exception rules which interact symbiotically with the default rules. The resulting model is therefore called a default hierarchy.

Credit assignment also leads to the idea of rule discovery. Rule discovery involves making changes in the agent's capabilities, replacing parts assigned low credit with new options. Rule discovery is in essence the generation of plausible hypotheses. It centers around the use of tested building blocks. Building blocks are combined to form new rules and these new rules are tested for their usefulness via their respective additive (or subtractive) affect on the agent's fitness. Past experience is directly incorporated, but innovation has broad latitude. This is how the agent learns, by taking risks and thinking outside of the box. The tag mechanism is used extensively in the rule discovery process.

At the Center for Naval Analysis, Andrew Ilachinski expanded on Holland's ideas about complex adaptive systems and modeled the actions of marine combatant units as a complex adaptive system. To Ilachinski, all the basic elements of a complex adaptive system were present in the model he was proposing. He demonstrated this as follows:

- nonlinear interaction – combat forces are composed of a large number of nonlinearly interacting parts including:
    - feedback loops in the C2 hierarchy;
    - interpretation of (and adaptation to) enemy actions;
    - decision-making processes;
    - elements of chance;

- nonreductionist – the fighting ability of a combat force cannot be understood as a simple aggregate function of the fighting ability of individual combatants;

- emergent behavior – the global patterns of behavior on the combat battlefield unfold, or emerge, out of nested sequences of local interaction rules and doctrine;

- hierarchical structure – combat forces are typically organized in a command and control (fractal-like) hierarchy;

- decentralized control – there is no master 'oracle' dictating the actions of each and every combatant and the course of battle is ultimately dictated by local decisions made by each combatant;

- self-organization – local action, which often appears 'chaotic', induces long-range order;

- nonequilibrium order – military conflicts, by their nature, proceed far from equilibrium and understanding how combat unfolds is more important than knowing the end state of the battle;

- adaptation – in order to survive, combat forces must continually adapt to a changing environment, and continually adapt to their enemy's adaptations;

- collective dynamics – there is a continual feedback between the behavior of (low-level) combatants and the (high-level) command structure.

Ilachinski points out that "agent-based simulations of complex adaptive systems are predicated on the idea that the global behavior of a complex system derives entirely from the low-level interactions among its constituent agents" (Ilachinski, 1997:3). Therefore lessons about the real-world system that an agent-based simulation is designed to model can be learned by looking at the emergent structures induced by the interaction processes taking place within the simulation.

Ilachinski stresses that the fundamental building block of most models of complex adaptive systems is the adaptive autonomous agent. These adaptive autonomous agents try to satisfy a set of goals in an unpredictable and changing environment. The agents are 'adaptive' in the sense that they can use their experience to continually improve their ability to deal with shifting goals and motivations. They are 'autonomous' in that they operate completely autonomously, and do not need to obey instructions issued by a God-like oracle.

According to Ilachinski, all adaptive autonomous agents are characterized by the same set of properties. Each agent is an entity that, by sensing and acting upon its environment, tries to fulfill a set of goals in a complex, dynamic setting. The agent can sense the environment through its sensors and act on the environment through its actuators. It has an internal information processing and decision-making capability that allows it to adapt to its locale. The agents anticipation of future states and possibilities, based on internal models (which are often incomplete and/or incorrect), often significantly alter the aggregate behavior of the system of which the agent is a part. Each agent possesses a set of goals which motivates its behaviors. These goals can take on diverse forms, such as desired local states, desired end goals, selective rewards to maximize, or internal needs (or motivations) that need to be kept within desired bounds.

The environment in which an agent thrives is only a small part of the actual system that is created when a complex adaptive systems model is built. Since a major component of an agent's environment consists of the other agents in the environment, agents generally spend a great deal of their time adapting to the adaptation patterns of the other agents around them. This leads to very diverse behaviors dependent on only small changes in one agent's behaviors. This sensitivity to initial conditions is a trademark of chaotic behavior as discussed above. The ability of an agent in a complex adaptive system to survive and evolve in a constantly changing environment is determined by its ability to continually find (either by chance, experience, or a combination of both) insightful new strategies to increase its overall fitness. Therefore, each agent must possess both a memory and an internal anticipatory mechanism that it uses to select the optimal tactic and/or strategy from among a set of predicted outcomes. Except for some very basic doctrine or rules of the game, the actual tactics for survival are not hard-wired into the agent. It is left to the agent to discover what works best for it. In the end, agent-based models are designed to look at the processes that occur due to agent interaction with the hope of figuring out what drives the process. Specific predictions derived from agent-based models are not feasible, but just like in chaos theory, indications of long-term trends (i.e. attractors) are very predictable and insightful.

A.2.3. The Uses and Applications of Complex Adaptive Systems Theory

Just like chaos theory has many uses in the real world, agent-based models have many real-world applications. Basically any system that derives most of its behavior from the interactions of the agents in that system can be modeled by a complex adaptive system. Investors in the stock market, the process of evolution, ants in an ant-hill, and combatants on the battlefield are just a few examples. Ilachinski took the last example, combatants on the battlefield, and created his ISAAC program. ISAAC—Irreducible Semi-Autonomous Adaptive Combat—took the perspective of agents as individual marine combat units whose autonomous actions on the battlefield were best modeled as a complex adaptive system. What ISAAC shows is how two opposing forces react (i.e. fight) when different control knobs on their environment are changed. ISAAC also allows the modeler to vary the type of marine combatant being modeled, ranging from an all-powerful strict follower of orders to the more cagey, rule-breaker. With each model run, trends in the outcome of a simulated battle are displayed and insight can be gleamed about the effects of different command and control strategies.

In the commercial world, many simulation oriented games use the concept of complex adaptive systems in order to drive the game's behavior. The SimCity game series by Will Wright of Maxis Software uses a form of complex adaptive system theory as a means to allow players to create worlds and then perturb their worlds with natural disasters and other calamities. The goal of the game is to build a world robust enough to adapt and overcome all the hazards thrown its way.

In another arena of complex adaptive systems theory is the notion of artificial life. As Ilachinski points out:

> The underlying supposition is that life owes at least as much to its existence to the way in which information is organized as it does to the physical substance (matter) that embodies that information. The fundamental concept of artificial life is emergence or the appearance of higher-level properties and behaviors of a system that, while obviously originating from the collective dynamics of that system's components, are neither to be found in nor are directly deducible from the lower-level properties of that system. (Ilachinski, 1997:7)

The remarkable characteristic of artificial life is that emergent properties are properties of the 'whole' that are not possessed by any of the individual parts making up the whole. Artificial life studies life by using artificial components to capture the behavioral essence of living systems. This bottom-up synthesis approach to modeling stands in marked contrast to the more conventional top-down analytical approach. Under a lot of circumstances, though, the bottom-up approach is much more robust and allows for more creative solutions.

In his report, Ilachinski points out that artificial life simulations are characterized by five general properties:

- they are defined by populations of simple programs or instructions about how individual parts all interact;
- there is no single 'master oracle' program that directs the action of all other programs;
- each program defines how simple entities respond to their environment locally;
- there are no rules that direct the global behavior of the system;
- behaviors on levels higher than individual programs are emergent and not explicitly programmed into the simulation.

Ilachinski claims that:

> Fundamentally, the artificial life approach represents a shift in focus from "hard-wiring" into a model a sufficient number of details of a system to yield a desired set of "realistic" behaviors to looking for universal patterns of high-level behavior that naturally and spontaneously emerge form an underlying set of low-level interactions and constraints. (Ilachinski, 1997:8)

A.2.4. The Use of Exploratory Modeling and Complex Adaptive Systems Models

Once a complex adaptive system model has been built, its best use for predictive purposes involves playing a lot of 'what-if' type games. In doing so, the modeler can test out hypotheses and see what the resulting system behavior is. This involves the use of exploratory modeling, a concept pioneered by Steve Bankes. Exploratory modeling is based on the notion of creating an ensemble of models that can then be used to answer specific questions. By running many different complex adaptive systems models with different parameter values, such an ensemble is created providing the decision maker a set of answers to different scenarios. The decision maker also has a means to compare the results of alternative simulation models to existing data, and therefore is able to verify the validity of the model. Since complex adaptive systems models require only knowledge about the actual autonomous behavior of each individual agent, exploratory modeling can exploit the prescriptive capabilities of models based on limited information. A complex system is modeled using simple agents. The emergent behavior of the simple agents is used to answer a plethora of 'what-if' questions. Simplicity leads to complexity which leads to prescriptive forecasting ability.

A.3. Artificial Life

A.3.1. Defining 'Life'

What is life? The apparent simplicity of this question has led to many heated debates within the science industry. Biologists stick to the notion that life is only a characteristic of organic matter, while computer scientists feel that limiting life-like qualities to organic entities is not fair. They claim that non-organic creations, like computer program based organisms, can be created as forms of life. But, the exact definition of 'life' still eludes most.

For centuries, a definition of life has been sought by scientists. During most of this time, biologists ruled and felt that any definition of life must be grounded in organic principles. The early searchers felt that living organisms had what they called a 'vital force' or 'élan vital' that supposedly existed only in living organisms. Some even considered it the divine component of life. As Steven Levy points out in his book Artificial Life: A Report from the Frontier where Computers meet Biology, the vitalists' ideals are still very present today:

> Though it would be a difficult task to unearth a modern scientist subscribing to that true doctrine [the presence of the vital force], vitalism of a sort seems to persist. There remains in us all an atavistic tendency to surrender biological prerogatives to any so-called beings outside the known family of earthbound organisms. There is a particular reluctance to concede the honor of life-form to anything created synthetically. This reluctance often transforms itself into profound skepticism, even mockery, when one suggests that life could be fashioned in a laboratory or in a computer, by using as the main substrate not organic molecules or other familiar forms of chemistry but something quite different—information. Information. The premise being that the basis of life is information, steeped in a dynamical system complex enough to reproduce and to bear offspring more complex than the parent. (Levy, 1992:22)

Aside from the entire organic argument, the remaining characteristics of life are also still debated. Many have suggested that among other characteristics, anything that is considered alive must be able to grow (and nourish itself), reproduce (and therefore procreate itself), adapt to its environment, and evolve to a higher level of fitness. In order to accomplish all this, the entity given the quality of life must be quite complex. This complexity issue is what has kept us from truly understanding life…until now. With the dawn of the computer revolution and the prolific rise of personal computer use, even the researcher with the tightest budget can afford to purchase or lease enough computing power to harness the massive processing ability needed to create complex organisms. The computer has opened the door for all to examine life more closely. We now have the means to uncover once and for all what it takes for something to be alive.

Christopher Langton, considered by most to be the father of the artificial life movement, showed that the stuff of life is not stuff, but it's the organization of that stuff that matters: "The leap you have to make is to think about machineness as being the logic of organization…It's not the material. There's nothing implicit about the material of anything—if you can capture its logical organization in some other medium you can have the same 'machine' because it's the organization that constitutes the machine, not the stuff it's made of" (quoted in Levy, 1992:117). To Langton, life was a process, not a collection of matter. And just like any other process, if the process of life could be duplicated, then artificial life is created, regardless of the materials used in that creative effort. According to Langton, life was defined as "a property of the organization of matter, rather than a property of the matter which is so organized" (quoted in Levy, 1992:118). In his opinion, there should be nothing in the definition of life precluding it from being created in non-organic forms. He went so far as to state that life did not even require a physical body as long as the processes peculiar to life were faithfully carried out by the entity in question.

The notion of life not being limited to organic material opened up the doors for the computer scientists to compete with the biologists and chemists in the race to see who could first create life. But, in the end, for either side to win the race, the two groups would find that their pursuits had a lot more in common than they dared admit to each other. The biologists and computer scientists would soon come to realize that, in order to create life, they would first have to understand life.

Stuart Kauffman, a physician by training but a philosopher by vocation, spent years in pursuit of artificial life from the biological standpoint until he came to the realization that the best way to create life was to use complex adaptive systems theory and the power of the modern computer. He found that logic, not matter, is what life was all about:

> Artificial life... is a way of exploring how complex systems can exhibit self-organization, adaptation, evolution, co-evolution, metabolism, all sorts of stuff. It is mimic of biology, although biologists don't know it yet. Out of it will emerge some sort of strange companion theory to biology... a particular substantiation of how living things work. This emerging discipline may be getting at what the logical structure is for living things. (quoted in Levy, 1992:124-125)

In his book At Home in the Universe, Kauffman professed support for the notion that life was an emergent property of the interacting parts of a system:

> Life... is not to be located in its parts, but in the collective emergent properties of the whole they create. Although life as an emergent phenomenon may be profound, its fundamental holism and emergence are not at all mysterious. A set of molecules either does or does not have the property that it is able to catalyze its own formation and reproduction from some simple food molecules. No vital force or extra substance is present in the emergent, self-reproducing whole. But the collective system does possess a stunning property not possessed by any of its parts. It is able to reproduce itself and to evolve. The collective system is alive. Its parts are just chemicals. (Kauffman, 1995:24)

To Kauffman, life, and artificial life for that matter, emerges from the dynamic behavior of a group of organisms, not from the individual organisms acting alone by themselves. The system, not what made it up, was what possessed the quality of life. To create artificial life, the system would have to become alive.

The implication of artificial life for biology could be truly profound. Kauffman was sure that if scientists could identify the forces of self-organization and complexity inherent in nature, then life-like behavior could be duplicated by man and computer. This duplicate of life would take advantage of the self-organizing forces found in nature and use them to structure man-made organisms to mimic their natural counterparts. In so doing, it would be possible to create life in the form of new organisms that harnessed the complexity of nature in their very essence. The emergent creations would be more complex than any human designer could create on his own, and this alone would be proof of the re-creation of life. With this accomplished, the implications of artificial life would be tremendous to say the least.

A.3.2. The History and Development of Artificial Life

Although Chris Langton is credited by most as the father of the artificial life movement thanks to his sponsorship of the first artificial life conference in September of 1987 at the Santa Fe Institute in New Mexico, it was actually John Von Neumann who came up with the first ideas related to artificial life back in the 1940s and 50s. Captivated by the complexity of nature around him:

> Von Neumann would readily admit that biological organisms were complex, more complicated than any artificial structure man had ever pondered. But ultimately, because he believed life was based on logic, he believed that we were capable of forcing organisms to surrender their secrets. (Levy, 1992:15)

To say Von Neumann was a genius is an understatement. Although he died nearly 50 years ago, principles he came up with are still widely used in the sciences today. Among them are his ideas about cellular automata. The automaton was the basis for what he considered the hope of creating artificial life. Von Neumann defined automata as:

> self-operating machines, specifically any such machine whose behavior could be unerringly defined in mathematical terms. An automaton is a machine that processes information, proceeding logically, inexorably performing its next action after applying data received from outside itself in light of instructions programmed within

> itself. (Levy, 1992:15)

Unfortunately for the science community, Von Neumann died before he could finish his research into automata and artificial life. The world lost perhaps one of its greatest thinkers when Von Neumann died, and the artificial life movement lost one of its greatest proponents. As a result, it would be decades before the pursuit of artificial life would again be restarted.

It wasn't until the late 1960s that artificial life once again came into the sciences. At Cambridge University, in 1968, the artificial life movement was reborn, and with a vengeance. John Horton Conway, a gifted mathematician with a knack for games, decided to create a game of his own—the game of Life. In his own words:

> Life occurs on a virtual checkerboard. The squares are called cells. They are in one of two states: alive or dead. Each cell has eight possible neighbors, the cells which touch its sides or its corners. If a cell on the checkerboard is alive, it will survive in the next time step (or generation) if there are either two or three neighbors also alive. It will die of overcrowding if there are more than three live neighbors, and it will die of exposure if there are fewer than two. If a cell on the checkerboard is dead, it will remain dead in the next generation unless exactly three of its eight neighbors are alive. In that case, the cell will be "born" in the next generation. (quoted in Levy, 1992:52)

That was it. The entire game of Life could be explained by those simple rules. However, playing the game of life did not result in simple outcomes. On the contrary, the game of life evolved into a plethora of incredibly interesting and complicated geometrical shapes.

Designed originally for a large game board, game of Life players began to create computer programs to play the game more easily. As a result even more complexity was discovered. Things like gliders that would move continually across the playing field in a diagonal motion gave players the eerie feeling that their computers were coming alive. Glider guns were soon to follow, and now not only did the game of Life have moving entities in its environment, but also a means to create new entities. The implications for artificial life were astounding, and all of a sudden the pursuit of artificial life was back.

Conway's game of Life ignited the fire smoldering since Von Neumann's death decades ago. Now, artificial life scientists were cropping up all over the country. With the computer revolution programmers soon came to realize that they would be the ones on the forefront of the artificial life movement.

Von Neumann's cellular automata were rediscovered. Their ability to surprise their users is what made them so appealing to artificial life practitioners. Just like life surprises us continuously, cellular automata could surprise their programmers:

> In a sense, with cellular automata, researchers turned the keys [to building and understanding models] over to nature. Forces of self-organization, not yet understood, along with the particular artificial physics in the given experiment, would drive the system. Sometimes, the destination would be unexpected. That was the mystery and power of CAs. (Levy, 1992:60-61)

As Tommaso Toffoli so eloquently pointed out in his graduate thesis on the worth of cellular automata (CA):

> The importance of cellular automata lies in their connection with the physical world. Particularly that of complex dynamical systems, where behavior arises as an emergent property of a number of variable forces. Because, unlike so many things simulated on the computer, CAs do not merely reflect reality—they are reality. They are actual dynamical systems. While they can be used to model certain physical systems, with the validity of each model to be determined by how well the results match the original, they can also be used to understand complex systems in general. (Levy, 1992:61)

Therefore, the use of cellular automata, Von Neumann's prophetic creations, would not only restart the entire artificial life movement, but would eventually lead to other sciences (like biology, chemistry, and physics) beginning to consider the implications of artificial life in their research.

Throughout the rest of the 1970s and the early 1980s, artificial life practitioners were hindered by the fact that the computer was still not user friendly enough. However, undaunted, people like Ed Fredkin and Stephen Wolfram pushed on and learned to use the bigger and more powerful university mainframe computers for their studies. These men expanded on Von Neumann's cellular automata ideas and began to shape the science of artificial life. Ed Fredkin went so far as to claim that life as we know it was just one big cellular automata. He postulated that living organisms in our universe acted by the same 'simple' (relatively so at least) rules that were characteristic of the organisms in a cellular automata environment and that we just didn't know or completely understand the rules…yet.

Stephen Wolfram sought to use cellular automata to make significant statements about the entire science of complex adaptive systems theory. He sought to use cellular automata to explain how complexity emerged from simple origins. Using one-dimensional cellular automata, he came up with what he called his four classes of cellular automata. Class 1 consisted of stable forms, those cellular automata that were all blank (representing all dead cells) or all dark (representing all living cells). Class 2 contained the periodic forms whose initial activity ceased and became stable (and very predictable) over time. Class 3 included all the chaotic forms where disorder reigned. Finally, class 4 was where the interesting cellular automata emerged. This class was characterized by disorder, yet massively complex behavior, very similar to real-life biological behavior, seemed to emerge. The richly complex forms that comprised this class, Wolfram felt, was where one would find life. Using cellular automata, one could recreate life.

Working off Wolfram's ideas, Chris Langton saw an apparent continuum of activity in the artificial life realm. Wolfram's four classes, he felt, described all manner of cellular automata that could be created, but he chose a more graphic way to display it (see Figure 10: Langton's Continuum of Activity). This figure shows Langton's view of

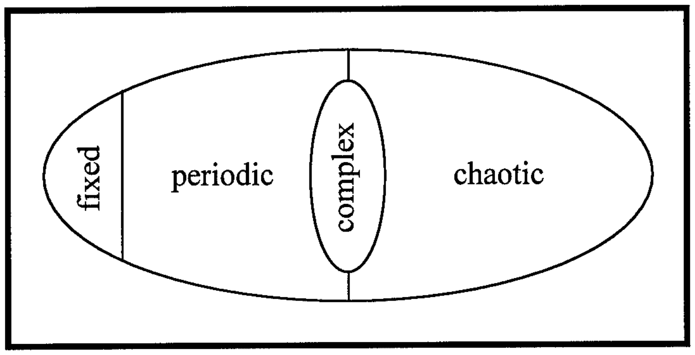


Figure 10: Langton's Continuum of Activity

the movement of information in complex adaptive systems. At the left, in the fixed region, is the area where information is frozen. Life cannot live there. To the right of it is a somewhat more flexible arena of limited periodic movement. The periodic nature of this movement, however, precludes life from living here too. To the far right, in the realm of chaos, there is no order and structure cannot be maintained, hence life is also doomed here. In the center, though, there exists a sweet spot that can maintain life. Here, information is stable enough to support structure but loose enough to allow for growth and change. This is where life lives. (Graphic from Levy, 1992:110).

Building on this continuum, Langton sought to isolate a general quality of life that could help determine whether or not an environment could support life. His dissertation work at the University of Michigan's Logic of Computers Group did just that. Langton was able to identify a control knob, what he called the lambda (λ) factor, that could be used to quantifiably express whether or not an environment could support life. Langton scaled lambda so that it ranged between 0 and 1. As described by Levy:

> If the λ value was very low, approaching zero, this represented a regime in which information was frozen. It could easily be retained through time, but it could not move. If you were to picture a substrate that illustrated this, you might choose a solid such as ice. Ice molecules retained their position and state as time passed, but, since they did not move around, no new information emerged. Under those circumstances, life could not thrive. If the λ value was very high, approaching its maximum value, then information moved very freely and chaotically and was very difficult to retain. The information could be envisioned as molecules in a gaseous, or vaporous, stage. Life could not be supported in these conditions, either. On the other hand, there was a certain area where information changed but not so rapidly that it lost all connection to where it had just been previously. This was akin to a liquid state. Langton discovered that it was the liquid regime that supported the most engaging events, those that would support the kind of complexity that was the mark of living systems. (Levy, 1992:109)

As shown in Figure 11: Langton's λ Factor (graphic from Levy, 1992:111), life was

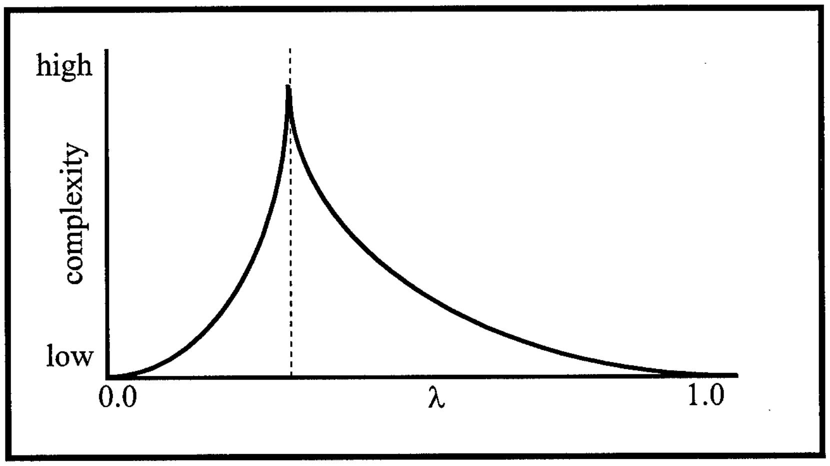


Figure 11: Langton's λ Factor

only supported in a very specific type of environment. This graph shows that a key ingredient to life was the proper mixture of order and chaos. As depicted in the figure, this proper mixture was on the edge of a cliff side. To the left of the cliff was the barren wasteland of stagnant information, where nothing could move. To the right lay chaos, were all order was lost and information flowed too freely. In the middle of all this, however, life could take advantage of both extremes and sustain itself. The order from the stagnant side helped counteract the disorder from the chaotic side, while the chaotic side helped the stagnant side enjoy some mobility. Langton claimed that scientists would find life "on the edge of chaos" (quoted in Levy, 1992:111).

Having now determined what environments could support life, Langton decided to begin codifying an analytical approach to implementing artificial life. He called his idea the bottom-up approach, espousing to the fact that small, agent-based models could be built where life-like qualities could emerge from the agents interactions with each other and their environment. To Langton, "the concept described the collective power of small actions rippling upward, combining with other small actions, often recursively so that action would beget reaction—until a recognizable pattern of global behavior emerged" (Levy, 1992:105). No central intelligence controlled the agents, they were in charge of their own destinies.

Langton later codified these elements of bottom-up architecture into his approach for designing the essential characteristics of a computer-base artificial life model. To him, artificial life models should:

- consist of populations of simple programs or specifications;
- have no single program that directs all of the other programs;

- each program details the way in which a simple entity reacts to local situations;
- there are no rules in the system that dictate global behavior;
- any behavior at levels higher than the individual programs emerges.

With the groundwork in theory now laid firmly in place, it came time to experiment with the new paradigm. Models began being built all over the scientific community and it was beginning to look like artificial life might have found its place in the realms of science. Applications in the computer sciences were not the only place artificial life mechanisms were springing up. Biologists, chemists, physicists, even graphical animators were using the tenets of artificial life to make newer and better models. It seemed as if the quest for artificial life was truly in full swing.

A.3.3. The Uses and Applications of Artificial Life

With the theory now firmly in place, artificial life practitioners were ready to try their new found science out. Budding computer scientists began experimenting with cellular automata and other simple organisms in the virtual world of the computer environment. There they could create their little life forms, set up an environment for them to live in, and then let them loose to live at large. Although very simplistic by today's standards of computer programming, these early artificial life forms were truly remarkable. The work done by these patriarchs of artificial life helped set the stage for future artificial life researchers in the decades to come.

By the 1980s, the artificial life movement had spread throughout the science community. Artificial life methods were being investigated in everything from biology and chemistry to physics and psychology. The advent of the personal computer also made it possible for what Steven Levy likes to call "garage band" scientists to begin tinkering with artificial life (Levy, 1992:85). Among these garage band scientists was a Craig Reynolds, and his work would forever change the world's outlook on artificial life.

A computer animator in the graphics division of the Symbolics Computer Company, Reynolds was destined to exploit the use of artificial life. Known for going on long walks during his lunch hour, he was constantly caught gazing at a flock of black birds that inhabited the park within which he liked to walk. These birds were what led Reynolds to begin his quest for artificial life.

To Reynolds, the flocking behavior exhibited by the birds was truly captivating. He noticed that there was no lead bird, no bird that directed the movements of the other birds in the flock. The flocking activity of all the birds was largely a decentralized phenomenon, and each bird seemed to follow some simple rules. The flocking behavior just emerged from the collective action of all the birds. In his own words:

> The motion of a flock of birds is...simple in concept yet is so visually complex it seems randomly arrayed and yet is magnificently synchronous. Perhaps most puzzling is the strong impression of intentional centralized control... Yet all evidence indicates that flock motion must be merely the aggregate result of the actions of individual animals, each acting solely on the basis of its local perception of the world. (quoted in Levy, 1992:76)

This apparent paradox, the seemingly unified flocking activity of the decentralized behaving birds, intrigued Reynolds. Here was the complexity of nature at its best. The simple minded birds were concerned only about their own immediate environment, but taken together, the entire group of birds' individual actions led to incredibly complex and dynamic group behavior. The flock was born of the individual birds acting on their own behalf but combining their actions to form one entire system. This was the essence of artificial life.

Reynolds began to observe the birds more critically. He soon boiled their behavior down to three simple rules (presented in Levy p, 1992:77):

1. a clumping force that kept the flock together;
2. an ability to match velocity so that birds in the flock would move at the same speed;
3. a separation force that prevented birds from getting too close to each other.

That was it. These three rules explained the entire system. There was nothing else going on in the flock except scores of birds following these three simple rules. If birds could do it, then Reynolds felt he could reproduce the same behavior using computer graphics.

Reynolds began working immediately on creating an artificially living flock of birds on a computer. Dubbing his creation 'boids', short for 'birdoids', he created an artificial flock using computer graphics and the three rules the birds in the real flock seemed to follow. His project was a success, and as he states, "as they flew, the boids would notice what their neighbors were doing—as though they were cells in a cellular automata—and apply that information to their own actions in the next time step" (quoted in Levy, 1992:77).

The notion of allelomimesis, a phrase coined by Jean-Louis Deneubourg, was created. Allelomimesis was based on the idea that an individual's actions were dictated by the actions of a neighbor. Allelomimesis was what controlled the flocking behavior of birds both in nature and Reynolds' boids simulation. Similar parallels could be drawn to swarms of flies, schools of fish, herd of gazelle, the list goes on. The scientific implications of this simple experiment were beginning to grow. In a way the artificial life movement was taking on a life of its own.

What Reynolds had accomplished was truly a break-through for the entire artificial life cause. He had successfully duplicated a natural phenomenon using artificial means. On his computer he had created artificial life that completely mimicked the real thing. As Steven Levy points out:

> Something was happening in this simulation, quite possibly something that had relevance to real birds, perhaps to group motion in general. It did not mean that birds actually used those rules, but it at least showed that using those rules would produce a behavior that looked like flocking. Would not ornithologists benefit from using these models to perform the sorts of experiments that would be impossible in the field but absurdly simple in the computer room? (Levy, 1992:79-80)

There was the key to artificial life's appeal. By creating an artificial life representation of the real system, experiments could be conducted in the virtual world that would be either too risky, too expensive, or just plain impossible in the real world. The benefits of artificial life were beginning to become very apparent. But there was even more.

The Mobile Robot (Mobot) Group at MIT's Artificial Intelligence Laboratory was becoming increasingly aware and very interested in the whole artificial life ideal. Attempting to create robots able to act on their own, the members of the Mobot group were becoming painfully aware of the current limits in the artificial intelligence community. They saw the artificial life movement as a solution to this dilemma and began pursuing it as a means to provide their robots autonomy.

Rodney Brooks, leader of the Mobot Group in the mid 1980s, was especially frustrated with the artificial intelligence paradigm then used for robotic intelligence. He felt that trying to get robots to think like humans was fruitless, especially since most robotic actions only required the intelligence of an insect at best. Instead of creating geniuses that could beat a chess grand champion but not navigate across a cluttered room because they spent too much time analyzing the terrain, he wanted to create insect like robots that were not particularly smart from humanity's standpoint, but that could very easily traverse even the toughest terrain just like real insects could. As such, ethology, or the study of animal behavior, became an integral part of a roboticists knowledge base.

Using these new artificial life ideas, the researchers at the Mobot Group began creating real artificial life. Their robots resembled the intelligence they were trying to mimic, namely insects. Built with short stout legs and the ability to move them all in an insect like rhythm, these little machines were beginning to act more and more like the insects they drew their design from. Creations like Genghis and Attila became over-night sensations. Robots were finally being built that could actually do something useful. Their construction was such that they looked and acted like metallic cockroaches. They responded to their environment by following a simple set of rules within their data-base brain, and then constantly adapting themselves to their new surroundings.

NASA became particularly interested in these new robotic creations. If humanity sought to investigate the other planets in the solar system, it would need to send out some explorers to collect data first. Robots were the natural choice for such a mission. However, control from Earth would be difficult, especially as the distances grew. Sending information back and forth to a robot on Mars about what foot to place in front of the other would take minutes and be subject to signal degradation. It would make more sense for the robots to walk and perform these simple functions independently and rely on Earth control only for major mission level objectives. In short, robots that were 'alive' and could implement their own solutions to problems would fare much better than those that relied on continued support from a million miles away. Robots based on artificial life became NASA's primary vehicle for interstellar exploration. As-a-matter-of-fact, the robot used to explore the Martian landscape just recently can thank its ancestors Genghis and Attila for laying down the groundwork for its ability to maneuver the treacherous landscape on Mars.

Lately, though, the most startling advances in the artificial life movement have come by some rather dubious means. Experimenting with parasitic computer code in late 1983, Fred Cohen created the first computer virus on the 3rd of November that year. Hoping to use his virus to perform routine banal tasks relating to file maintenance, Cohen truly did not foresee what his Frankenstinian creation would beget. Realizing that his virus was supposed to "infect other programs by modifying them to include a possibly evolved copy of itself" (quote by Cohen in Levy, 1992:312), he failed to see the diabolical nature by which such a program could be used. Instead, Cohen felt that good viruses could "lead to remarkable enhancements of information systems, which should come to bear the brunt of the world's menial work" (quoted in Levy, 1992:333). However, viruses were too easy to create and too easy to use for destructive purposes, and Cohen didn't realize all this until it was too late. Once created, the virus revolution took off, and today millions of computer systems are infected by malicious descendants of Cohen's first creation.

The creation of computer viruses, though, did create quite a stir in the artificial life movement. The debate over life's definition was started all over again. If humans could create viruses on computers, had artificial life really been created. As Steven Levy points out, the answer was not forthcoming:

> It was true that Cohen's virus, the Brain virus [written by two Pakistani computer store owners in 1986], certain Core War organisms [created by A. K. (Kee) Dewdney in May of 1984)], and hundreds of other persistent computer creatures shared a frightening quality with natural organisms. All drew on forces much more powerful than themselves, in a manner consistent with life's slickest move: an apparent violation of the second law of thermodynamics. As Leonard Adleman was quick to perceive, Cohen's virus accomplished this trick with eerie similarity to the devices of biological viruses. Yet this alone did not bestow them life. In fact, it was not clear whether a biological virus itself could be included in the society of life. The debate over this matter had long ago turned into a unproductive stalemate. Biological viruses are no more than naked strands of nucleic acid, either RNA or DNA, surrounded by a sheath of protein. They cannot perform their key organic functions, particularly reproduction, without commandeering the host cells they rudely violate. They remain dormant until invading a cell of a host species, whereupon they burst into activity, hijacking the mechanics of the cell so that it performs tasks geared toward the viruses' ends. The materials inside the cell are appropriated to reproduce new viruses. At times, even parts of the host cell's DNA code are reinterpreted to aid in the production of viruses. Thus viruses are incomplete organisms. To some, this incompleteness indicated that viruses were something less than fully alive. To others, it seemed obvious that viruses shared so much with organisms universally considered alive—the family of life, from bacteria and up the ladder of complexity—that the boundaries of our definition must include them. Because we had no definition of life, the question was perpetually up for grabs. (Levy, 1992:326-327)

Overall, computer viruses do make a rather compelling case for aliveness, at least to the degree that biological viruses can be considered alive. However, the debate does not end there. If computer viruses can be considered alive, then what about Reynold's boids, or Brook's robots. Would these too be considered alive? This debate still rages on in the halls of science.

Artificial life is an incredibly dynamic topic. As Levy shows, it seems to spit in the face of science's strict second law of thermodynamics:

> According to the second law of thermodynamics, as time passes, energy dissipates and becomes unusable. Order deteriorates. But life seems to behave as if it has not bothered to read the second law. Life seems to propagate order over time. From its unquestionably simpler beginnings, the history of life as we know it has been a trajectory of increasing well-ordered complexity. (Levy, 1992:34)

If this is true, and most artificial life practitioners believe it is, then how long will it be before humanity really does create artificial life? More importantly, how long before humanity creates artificial life to such a degree that we bring about our own extinction? When will artificial life surpass us in the evolutionary arms race?

APPENDIX B – JAVA SOURCE CODE

B.1. PilotRetention Class

```
// Capt Marty Gaupp's Air Force Institute of Technology Thesis Project
//
// last revised on 25 Feb 1999
//
// this is the main applet class that controls and calls all the other classes

import java.awt.*;
import java.applet.*;
import java.util.*;
import java.io.*;

// main applet class
//  this class extends Applet and controls the entire program

public class PilotRetention extends Applet implements Runnable
{
    // parameteres used throughout the program to control various aspects of execution

    static int DELAY = 10;                      // used as a delay in the printing of the summary statistics
                                                //  a counter is mod'ed by this number
                                                //  when the result is zero, a statistic is recorded
    static boolean LOOP = false;                // used to toggle the looping of the algorithm for LOOPLENGTH/10 observations
    static int ITERATIONS = 5;                  // the number of iterations to perform for each model
    static int LOOPLENGTH = 4000;               // LOOPLENGTH/DELAY is the number of observations that will be taken
    static boolean FILEOUT = LOOP;              // used to toggle the output of data to a file
    static String outputFileName;               // string to hold the name of the output file
    static FileOutputStream dataOut = null;     // file output stream for the data
    static int INITMONEYRHO = 55;               // initial value for the money rho
    static int INITTIMERHO = 25;                // initial value for the time rho
    static int INITHIRINGVALUE = 80;            // initial value for the hiring value
    static int INITPAYGAPVALUE = 80;            // initial value for the paygap value
    static int INITOPSTEMPOVALUE = 80;          // initial value for the opstempo value
    static int ROW = 0;                         // row counter variable for the OutputArray in the SummaryStats class
    static int COL = 0;                         // column counter variable for the OutputArray in the SummaryStats class


    // the applet's class-wide variables

    Thread runner;                              // thread of execution
    public PlayingField agentField;             // the playing field
    public MainCaption title;                   // title of the applet
    public EnvironmentControl environment;      // the environmental controls
    public PilotControl pcontrol;               // the pilot's utility curves
    public SummaryStats stats;                  // the summary statistics
    public Panel north;                         // top of the screen
    public Panel south;                         // bottom of the screen
    public Panel center;                        // center of the screen
    public Panel east;                          // east of the screen
    public Panel west;                          // west of the screen
    public int wdth;                            // width of the PlayingField
    public int hght;                            // height of the PlayingField

    // static variables used to hold values from other classes

    static public int hiringValue;              // the value of the hiring environmental control
    static public int paygapValue;              // the value of the pay gap environmental control
    static public int opstempoValue;            // the value of the operations tempo environmental control
    static public int tmwtValue;                // the value of the time/money weight environmental control
    static public double moneyWt;               // the weighting of the money utility curve
    static public double timeWt;                // the weighting of the time utility curve
    static public int moneyRho;                 // the rho value of the money utility curve
    static public int timeRho;                  // the rho value of the time utility curve

    // static variables for holding summary stats values

    static public int totalNumberExited = 0;    // the total number that exited the system
    static public int totalExitMin = 0;         // the total number that exited at the end of UPT commitment (8 YOS)
    static public int totalExitMid = 0;         // the total number that exited at the middle of their career (14 YOS)
    static public int totalExitEnd = 0;         // the total number that exited at the end of their career (20 YOS)

    // the applet's class-wide static variables

    static int appWdth = 700;                   // width of the applet's window
    static int appHght = 500;                   // height of the applet's window
    static Vector pilots;                       // pilots is a vector of all the pilots
    static int NUMAGENTS = 10;                  // keeps track of the number of agents
    static int SLEEP = 20;                      // the pause between animation repaints
    static int NUMNBORS = 2;                    // number of neighbors to keep track of
                                                //  increasing it increases the cohesiveness of the pack
```

```
static double KICK = 0.05;                     // percentage of time that a random kick is given to the system
                                               //  this gives the entire system a stochastic nature
static int REVRSDIST = 200;                    // used to indicate when the agent needs to reverse directions
                                               //  this indicates the agent's personal space
                                               //  get inside this and you should start moving in the opposite direction
static double ACCEL = 0.3;                     // agent's acceleration in pixels
static double ACCELTOMID = 0.2;                // agent's acceleration toward the middle of the playing field
                                               //  used to adjust the agent's position if he get's too far off center
static double ACCELTOSEP = 0.2;                // agent's acceleration toward the separation zone
static double ACCELTORET = 0.2;                // agent's acceleration toward the retention zone
static double MAXSPEED = 7.0;                  // upper limit for the agent's speed
static double BOUNCESPEED = 0.8;               // speed used to bounce off other agents and canvas boundaries
static double INITIALSPEED = 4.0;              // used to give instantiated agents their initial speed
static int SEPTHRESH = 100;                    // the system's separation threshold
                                               //  once this many separation requests have been lodged, the agent can separate

// this method initializes the applet
//  it sets up the entire graphic display via the border layout manager

public void init()
{

    resize(appWdth, appHght);

    setLayout(new BorderLayout());
    setBackground(Color.white);

    north = new Panel();
    north.setLayout(new FlowLayout());
    title = new MainCaption();
    north.add(title);

    south = new Panel();
    south.setLayout(new FlowLayout());
    stats = new SummaryStats();
    south.add(stats);

    center = new Panel();
    center.setLayout(new FlowLayout());
    agentField = new PlayingField();
    center.add(agentField);

    east = new Panel();
    east.setLayout(new FlowLayout());
    environment = new EnvironmentControl();
    east.add(environment, BorderLayout.CENTER);

    west = new Panel();
    west.setLayout(new FlowLayout());
    pcontrol = new PilotControl();
    west.add(pcontrol);

    add(north, BorderLayout.NORTH);
    add(south, BorderLayout.SOUTH);
    add(center, BorderLayout.CENTER);
    add(east, BorderLayout.EAST);
    add(west, BorderLayout.WEST);

    // initialize the pilot array of agents
    pilots = new Vector(NUMAGENTS);
    hght = agentField.size().height;
    wdth = agentField.size().width;
    for(int i=0;i<NUMAGENTS;i++)
    {
        Agent newPilot = new Agent(i,hght,wdth);
        pilots.addElement(newPilot);
    }
    randomize();
    // end pilot array initialization

} // end init() method

// this method starts the entire applet
//  it starts the applet's main thread of execution

public void start()
{

    runner = new Thread(this);
    runner.start();

} // end start() method

// this method runs the entire applet
//  it initializes the pilot array of agents
//  it randomizes the neighbors of the pilots in the array
//  it calls the randomize() method KICK% of the time
//  it sleeps the entire system for SLEEP nanoseconds in order to allow the animation to be seen
//  it repaints the entire graphic after each iteration
```

```
public void run()
{

    if (FILEOUT)                                  // setup the output data file
    {
        outputFileName = new String("DataOut.out");
        try
        {
            dataOut = new FileOutputStream (outputFileName);
        }
        catch(IOException e)
        {
            System.out.println("Error opening " + outputFileName);
        }
        String header = new String("money rho\ttime rho\thiring\tpaygap\topstempo");
        PrintStream headerPS = new PrintStream(dataOut);
        headerPS.println(header);
    } // end if(FILEOUT)

    // call the randomize() method KICK% of the time

    if(LOOP)                                      // do the algorithm for LOOPLENGTH/10 observations
    {
        int runNumber = 0;                        // keeps track of the run number
        COL = 0;                                  // reset the column number after each set of iterations for a given paramater set
        // get the original paramater values
            hiringValue = environment.getHiringValue();
            paygapValue = environment.getPaygapValue();
            opstempoValue = environment.getOpstempoValue();
            timeWt = ((double) (100 - environment.getTmwtValue()))/100;
            moneyWt = ((double) environment.getTmwtValue())/100;
            moneyRho = pcontrol.getMoneyRhoValue();
            timeRho = pcontrol.getTimeRhoValue();
        if (FILEOUT)                              // output the iteration data to a file
        {
            PrintStream iterPS = new PrintStream(dataOut);
            String dataLine = new String("money = " + moneyRho + "\t" +
                                         "time = " + timeRho + "\t" +
                                         "hire = " + (100-hiringValue) + "\t" +
                                         "gap = " + (100-paygapValue) + "\t" +
                                         "ops = " + (100-opstempoValue));
            iterPS.println(dataLine);
            PrintStream statsHeaderPS = new PrintStream(PilotRetention.dataOut);
            String statsHeader = new String("Min - 1\tMid - 1\tEnd - 1\t" +
                                            "Min - 2\tMid - 2\tEnd - 2\t" +
                                            "Min - 3\tMid - 3\tEnd - 3\t" +
                                            "Min - 4\tMid - 4\tEnd - 4\t" +
                                            "Min - 5\tMid - 5\tEnd - 5\t" +
                                            "Min - AVG\tMid - AVG\tEnd - AVG");
            statsHeaderPS.println(statsHeader);
        } // end if for iteration data to file output

        for (int iter=0;iter<ITERATIONS;iter++)
        {
            runNumber = runNumber + 1;
            System.out.println("Doing run number " + runNumber + " which is iteration number " + (iter+1) +
                               " with moneyRho = " + moneyRho + " timeRho = " + timeRho +
                               " hiringValue = " + (100-hiringValue) + " paygapValue = " + (100-paygapValue) +
                               " opstempoValue = " + (100-opstempoValue));
            int oldNUMAGENTS;
            int timer = 0;

            for (int l=0;l<LOOPLENGTH;l++)
            {
                oldNUMAGENTS = NUMAGENTS;
                if (Math.random() < KICK) randomize();

                // update values from environmental control

                hiringValue = environment.getHiringValue();
                paygapValue = environment.getPaygapValue();
                opstempoValue = environment.getOpstempoValue();
                timeWt = ((double) (100 - environment.getTmwtValue()))/100;
                moneyWt = ((double) environment.getTmwtValue())/100;

                // update values from utility curves

                moneyRho = pcontrol.getMoneyRhoValue();
                timeRho = pcontrol.getTimeRhoValue();

                // add agents every 100 passes through the loop

                timer = timer + 1;
                if (timer%100 == 0)
                {
                    int curSize = pilots.size();
                    int newNum = curSize;
                    Agent newPilot = new Agent(newNum,hght,wdth,0);
                    pilots.addElement(newPilot);
                    NUMAGENTS = NUMAGENTS + 1;
                    randomize();
                }
            // remove agents if exit is true

            for (int i=0;i<NUMAGENTS;i++)
            {
                Agent curPilot = (Agent) pilots.elementAt(i);
                if (curPilot.exit)
                {
                    totalNumberExited = totalNumberExited + 1;
                    if (curPilot.YOS < 12) totalExitMin = totalExitMin + 1;
                    if ((curPilot.YOS >= 12) && (curPilot.YOS < 20)) totalExitMid = totalExitMid + 1;
                    if (curPilot.YOS >= 20) totalExitEnd = totalExitEnd + 1;
                    pilots.removeElementAt(i);
                    NUMAGENTS = NUMAGENTS - 1;
                    randomize();
                }
            }
            agentField.repaint();
            stats.repaint();
            try
            {
                Thread.sleep(SLEEP);
            }
            catch(InterruptedException e)
            {
            }
        } // end LOOPLENGTH loop
        // increase the COL value by one
        COL = COL + 1;
        // destroy the old agents and create 10 new ones
        // destroy the old agents
        pilots.removeAllElements();
        NUMAGENTS = 10;
        // create 10 new agents
        for(int i=0;i<NUMAGENTS;i++)
        {
            Agent newPilot = new Agent(i,hght,wdth);
            pilots.addElement(newPilot);
        }
        randomize();
        totalNumberExited = 0;
        totalExitMin = 0;
        totalExitMid = 0;
        totalExitEnd = 0;
        if (COL > 0) ROW = 0;              // reset the ROW counter based on COL's value

    } // end ITERATIONS loop

} // end the algorithm for LOOPLENGTH/10 observations

else                                       // use the algorithm indefinitely
{

    int oldNUMAGENTS;
    int timer = 0;

    while (true)
    {
        oldNUMAGENTS = NUMAGENTS;
        if (Math.random() < KICK) randomize();

        // update values from environmental control

        hiringValue = environment.getHiringValue();
        paygapValue = environment.getPaygapValue();
        opstempoValue = environment.getOpstempoValue();
        timeWt = ((double) (100 - environment.getTmwtValue()))/100;
        moneyWt = ((double) environment.getTmwtValue())/100;

        // update values from utility curves

        moneyRho = pcontrol.getMoneyRhoValue();
        timeRho = pcontrol.getTimeRhoValue();

        // add agents every 100 passes through the loop

        timer = timer + 1;
        if (timer%100 == 0)
        {
            int curSize = pilots.size();
            int newNum = curSize;
            Agent newPilot = new Agent(newNum,hght,wdth,0);
            pilots.addElement(newPilot);
            NUMAGENTS = NUMAGENTS + 1;
            randomize();
        }

        // remove agents if exit is true
```

```
                for (int i=0;i<NUMAGENTS;i++)
                {
                    Agent curPilot = (Agent) pilots.elementAt(i);
                    if (curPilot.exit)
                    {
                        totalNumberExited = totalNumberExited + 1;
                        if (curPilot.YOS < 12) totalExitMin = totalExitMin + 1;
                        if ((curPilot.YOS >= 12) && (curPilot.YOS < 20)) totalExitMid = totalExitMid + 1;
                        if (curPilot.YOS >= 20) totalExitEnd = totalExitEnd + 1;
                        pilots.removeElementAt(i);
                        NUMAGENTS = NUMAGENTS - 1;
                        randomize();
                    }
                }

                agentField.repaint();
                stats.repaint();
                try
                {
                    Thread.sleep(SLEEP);
                }
                catch(InterruptedException e)
                {
                }
            }
        } // end the indefinite duration algorithm
    } // end run() method

    // this method randomizes the neighbors of each pilot
    //  it is used to add some randomness into the process

    public void randomize()
    {

        int n;

        // loop through all the pilots in the array
        for (int k=0;k<NUMAGENTS;k++)
        {
            Agent cur = (Agent) pilots.elementAt(k);
            // loop through all of cur's neighbors
            for (int j=0;j<cur.numNbors;j++)
            {
                n = (int) (Math.random()*NUMAGENTS);
                cur.neighbors[j] = (Agent) pilots.elementAt(n);     // set cur's current neighbor to be this
                                                                    //  randomly picked pilot
            }
        }
    } // end randomize() method

} // end class PilotRetention

// this class defines the PlayingField as an extension of Canvas
//  it then sets up the PlayingField's paint method

class PlayingField extends Canvas
{

    // constructor

    public PlayingField()
    {
        resize(400,325);
        setBackground(Color.black);
        setForeground(Color.white);
    } // end PlayingField constructor

    public void paint(Graphics g)
    {
        Agent pilot;

        // draw the caption
        g.setColor(Color.yellow);
        g.drawString("2Lt's - Yellow",5,318);
        g.setColor(Color.orange);
        g.drawString("1Lt's - Orange",80,318);
        g.setColor(Color.green);
        g.drawString("Capt's - Green",165,318);
        g.setColor(Color.blue);
        g.drawString("Maj's - Blue",250,318);
        g.setColor(Color.red);
        g.drawString("LtCol's - Red",320,318);

        // loop through each of the agents in the array pilots
        //  get that pilot's neighbors
        //  process that pilot to get his updated parameters
        //  draw that pilot on the PlayingField
        for (int i=0;i<PilotRetention.NUMAGENTS;i++)
        {
            pilot = (Agent) PilotRetention.pilots.elementAt(i); // set pilot equal to the current agent in pilots
            pilot.GetNeighbors();                               // get the pilot's neighbors
            pilot.UpdateFitness();                              // update the pilot's fitness function
            pilot.ProcessAgent();                               // process the pilot to get his updated parameters
            pilot.Draw(g);                                      // draw the pilot on the PlayingField

        }
    } // end paint(Graphics g) method

} // end class Playing Field

// this class sets up the caption at the top of the applet

class MainCaption extends Canvas
{

    // variables

    int indent = PilotRetention.appWdth/2 - 175;    // used to "center" the caption

    // constructor

    public MainCaption()
    {
        resize(PilotRetention.appWdth,20);
        setBackground(Color.white);
        setForeground(Color.black);
    } // end MainCaption constructor

    public void paint(Graphics g)
    {
        Font fontdef = new Font("Dialog", Font.BOLD, 15);
        g.setFont(fontdef);
        g.drawString("Pilot Inventory Complex Adaptive System (PICAS)",indent,15);
    } // end paint(Graphics g) method

} // end class MainCaption
```

B.2. Agent Class

```
// Capt Marty Gaupp's Air Force Institute of Technology Thesis Project
//
// last revised on 25 Feb 1999
//
// class to define and process Agents

import java.awt.*;

public class Agent
{
    // class wide variables

    int IDnum;                                  // the identification number of the agent (its index)
    Color agentColor;                           // the agent's color
    double YOS;                                 // the agent's Years Of Service
    boolean exit;                               // determines whether or not the agent should exit the system
                                                //  = true if the agent is about to exit
                                                //  = false if the agent isn't about to exit
    int sepCount;                               // counts the agent's separation requests
                                                //  a separation request is lodged everytime the agent passes
                                                //  through the lower-right hand (LRH) quadrant of the playing field
                                                //   this is the LRH quarter of the LRH quarter of the playing field
    int xPos;                                   // agent's current X position
    int yPos;                                   // agent's current Y position
    int xTail;                                  // agent's old X position (where the tail ends)
    int yTail;                                  // agent's old Y position (where the tail ends)
    double xSpeed;                              // agent's speed in the X direction
    double ySpeed;                              // agent's speed in the Y direction
    int canvasHt;                               // height of the canvas
    int canvasWt;                               // width of the canvas
    int numNbors;                               // number of neighbors to keep track of
                                                //  increasing it increases the cohesiveness of the pack
    int revrsdist;                              // used to indicate when the agent needs to reverse directions
                                                //  this indicates the agent's personal space
                                                //  get inside this and you should start moving in the opposite direction
    Agent neighbors[];                          // array of agent's neighbors (which are other agents)
    double accel;                               // agent's accelaration in pixels
    double accelToMid;                          // agent's acceleration toward the middle of the playing field
                                                //  used to adjust the agent's position if he get's too far off center
    double accelToSep;                          // agent's acceleration toward the separation zone
    double accelToRet;                          // agent's acceleration toward the retention zone
    double maxSpeed;                            // upper limit for the agent's speed
    double bounceSpeed;                         // speed used to bounce off other agents and canvas boundaries
    double initialSpeed;                        // used to give instantiated agents their initial speed
    double fitness;                             // agent's fitness function
                                                //  used to determine the happiness and propensity of the agent
    int sepThresh;                              // the system's separation threshold
                                                //  once this many separation requests have been lodged, the agent
can separate

    // temporary variables for the fitness calculations

    double moneyInput;
    double moneyLookUp;
    int rhoMoney;
    double wtMoney;
    double timeInput;
    double timeLookUp;
    int rhoTime;
    double wtTime;

    // Agent constructor method for initial random set of agents
    //  create the agents randomly
    //  randomly assign them:   a position - somewhere in the playing field
    //                          a YOS - uniformly distributed between 0 and 20
    //                          a fitness - 0.5 plus or minus 0.2


    public Agent(int agentNumber, int ht, int wdth)
    {
        // assign each agent it's passed values
        IDnum = agentNumber;
        canvasHt = ht;
        canvasWt = wdth;
        exit = false;
        sepCount = 0;

        // get the values for the static variables from the main PilotRetention applet
        numNbors = PilotRetention.NUMNBORS;
        revrsdist = PilotRetention.REVRSDIST;
        accel = PilotRetention.ACCEL;
        accelToMid = PilotRetention.ACCELTOMID;
        accelToSep = PilotRetention.ACCELTOSEP;
        accelToRet = PilotRetention.ACCELTORET;
        maxSpeed = PilotRetention.MAXSPEED;
        bounceSpeed = PilotRetention.BOUNCESPEED;
        initialSpeed = PilotRetention.INITIALSPEED;
        sepThresh = PilotRetention.SEPTHRESH;

        // give each agent it's individual parameters
        //  set the agent's initial position and velocity at instantiation
            xPos = (int)(wdth*Math.random());
            yPos = (int)(ht*Math.random());
            xSpeed = (initialSpeed*Math.random());
            ySpeed = (initialSpeed*Math.random());
        // assign a random YOS and color the agent accordingly
            YOS = (int)(20*Math.random());
            if (YOS < 2) agentColor = Color.yellow;
            if ((YOS >=2) && (YOS < 4)) agentColor = Color.orange;
            if ((YOS >=4) && (YOS < 12)) agentColor = Color.green;
            if ((YOS >=12) && (YOS < 16)) agentColor = Color.blue;
            if ((YOS >=16) && (YOS < 20)) agentColor = Color.red;
        // give each agent a fitness value
            fitness = 0.3 + (0.4*Math.random());
        xTail = xPos;
        yTail = yPos;
        neighbors = new Agent[numNbors];

    } // end Agent constructor initial random set of agents

    // Agent constructor method for new agent with 0 YOS

    public Agent(int agentNumber, int ht, int wdth, int years)
    {
        // assign each agent it's passed values
        IDnum = agentNumber;
        canvasHt = ht;
        canvasWt = wdth;
        YOS = years;
        exit = false;
        sepCount = 0;

        // get the values for the static variables from the main PilotRetention applet
        numNbors = PilotRetention.NUMNBORS;
        revrsdist = PilotRetention.REVRSDIST;
        accel = PilotRetention.ACCEL;
        accelToMid = PilotRetention.ACCELTOMID;
        accelToSep = PilotRetention.ACCELTOSEP;
        accelToRet = PilotRetention.ACCELTORET;
        maxSpeed = PilotRetention.MAXSPEED;
        bounceSpeed = PilotRetention.BOUNCESPEED;
        initialSpeed = PilotRetention.INITIALSPEED;
        sepThresh = PilotRetention.SEPTHRESH;

        // give each agent it's individual parameters
        //  set the agent's initial position and velocity at instantiation
            xPos = (int)(wdth*Math.random());
            yPos = (int)(ht*Math.random());
            xSpeed = (initialSpeed*Math.random());
            ySpeed = (initialSpeed*Math.random());
        // set up the agent's color depending on YOS
            if (YOS < 2) agentColor = Color.yellow;
            if ((YOS >=2) && (YOS < 4)) agentColor = Color.orange;
            if ((YOS >=4) && (YOS < 12)) agentColor = Color.green;
            if ((YOS >=12) && (YOS < 16)) agentColor = Color.blue;
            if ((YOS >=16) && (YOS < 20)) agentColor = Color.red;
        xTail = xPos;
        yTail = yPos;
        neighbors = new Agent[numNbors];

        // determine the agent's fitness
        //  moneyInput = average of hiringValue and paygapValue
        //  YOS affects money and time wts
        //   each YOS over 10 adds 0.01 to time and subtracts 0.01 from money
        //   each YOS under 10 subtracts 0.01 from time and adds 0.01 to money
        moneyInput = 0.5*(double)PilotRetention.hiringValue + 0.5*(double)PilotRetention.paygapValue;
        timeInput = (double)PilotRetention.opstempoValue;
        rhoMoney = PilotRetention.moneyRho;
        rhoTime = PilotRetention.timeRho;
        wtMoney = PilotRetention.moneyWt+(.01*(10-YOS));
            // ensure wtMoney isn't < 0 or > 1
            if (wtMoney > 1.0) wtMoney = 1.0;
            if (wtMoney < 0.0) wtMoney = 0.0;
        wtTime = PilotRetention.timeWt-(.01*(10-YOS));
            // ensure wtTime isn't < 0 or > 1
            if (wtTime > 1.0) wtTime = 1.0;
            if (wtTime < 0.0) wtTime = 0.0;
        // get the lookup values using utility curve theory
        if (rhoMoney == 0)
        {
            moneyLookUp = moneyInput/100;
        }
        if (rhoMoney !=0)
        {
            moneyLookUp = (1-Math.exp(-moneyInput/rhoMoney))/(1-Math.exp(-100.0/rhoMoney));
        }
    if (rhoTime == 0)
    {
        timeLookUp = timeInput/100;
    }
    if (rhoTime !=0)
    {
        timeLookUp = (1-Math.exp(-timeInput/rhoTime))/(1-Math.exp(-100.0/rhoTime));
    }
    // update the fitness of this agent
    fitness = 0.3 + (0.4*Math.random());
    double newFitness = wtMoney*moneyLookUp + wtTime*timeLookUp;
    double deltaFitness = fitness - newFitness;
    if (deltaFitness > 0) fitness = fitness - 0.5*deltaFitness;
    if (deltaFitness < 0) fitness = fitness - 0.5*deltaFitness;
    if(fitness < 0.0) fitness = 0.0;
    if(fitness > 1.0) fitness = 1.0;

    // error checking screen prints

} // end Agent constructor for new agent with 0 YOS


// this method calculates the distance between two agents using the distance formula
//  the distance between the current agent and the one passed to the method as a parameter is calculated

public int distance(Agent passedAgent)
{
    int dist;                                  // the distance calculated in this method
    int xDist;                                 // the distance in the x direction
    int yDist;                                 // the distance in the y direction
    xDist = xPos - passedAgent.xPos;
    yDist = yPos - passedAgent.yPos;
    dist = xDist*xDist + yDist*yDist;
    return (dist);
} // end distance(passedAgent) method

// this method is used to get the numNbors that are nearest to the agent
//  the current agents calls this method, therefore it has access to all of its own parameters
//  the neighbors are referenced by an indexed array

public void GetNeighbors()
{
    Agent currentNbor;

    // cycle through the neighbors of the agent that called this method to determine if
    //  the current neighbors are really the closest agents to the agent calling this method

    for (int i=0;i<numNbors;i++)
    {
        currentNbor = neighbors[i];            // these are the agent's current neighbors
        for (int j=0;j<numNbors;j++)
        {
            if (distance(currentNbor.neighbors[j])<distance(currentNbor))
            {
                // then the current agnet's neighbor's neighbor[j] is closer than the current
                //  agent's neighbor[i], so swap them and have the jth agent (which is closer)
                //  become the new neighbor
                neighbors[i] = currentNbor.neighbors[j];
            }
        }
    }
} // end GetNeighbors() method

// This method updates each agent's fitness
// note the following with respect to these calculations:
//  moneyInput = average of hiringValue and paygapValue
//  YOS affects money and time wts
//   each YOS over 10 adds 0.01 to time and subtracts 0.01 from money
//   each YOS under 10 subtracts 0.01 from time and adds 0.01 to money

public void UpdateFitness()
{
    double newFitness;
    double deltaFitness;
    // get some initial values
    moneyInput = 0.5*(double)PilotRetention.hiringValue + 0.5*(double)PilotRetention.paygapValue;
    timeInput = (double)PilotRetention.opstempoValue;
    rhoMoney = PilotRetention.moneyRho;
    rhoTime = PilotRetention.timeRho;
    wtMoney = PilotRetention.moneyWt+(.01*(10-YOS));
        // ensure wtMoney isn't < 0 or > 1
        if (wtMoney > 1.0) wtMoney = 1.0;
        if (wtMoney < 0.0) wtMoney = 0.0;
    wtTime = PilotRetention.timeWt-(.01*(10-YOS));
        // ensure wtTime isn't < 0 or > 1
        if (wtTime > 1.0) wtTime = 1.0;
        if (wtTime < 0.0) wtTime = 0.0;
    // get the lookup values using utility curve theory
    if (rhoMoney == 0)
    {
        moneyLookUp = moneyInput/100;
    }
    if (rhoMoney !=0)
    {
        moneyLookUp = (1-Math.exp(-moneyInput/rhoMoney))/(1-Math.exp(-100.0/rhoMoney));
    }
    if (rhoTime == 0)
    {
        timeLookUp = timeInput/100;
    }
    if (rhoTime !=0)
    {
        timeLookUp = (1-Math.exp(-timeInput/rhoTime))/(1-Math.exp(-100.0/rhoTime));
    }
    // update the fitness of this agent
    newFitness = wtMoney*moneyLookUp + wtTime*timeLookUp;
    deltaFitness = fitness - newFitness;
    if (deltaFitness > 0) fitness = fitness - 0.01*deltaFitness;
    if (deltaFitness < 0) fitness = fitness - 0.01*deltaFitness;
    if(fitness < 0.0) fitness = 0.0;
    if(fitness > 1.0) fitness = 1.0;

    // error checking screen prints

} // end UpdateFitness() method

public void ProcessAgent()
{
    // declare local method variables

    int revrs;                              // used to alter the accel parameter (it acts as a multiplier
                                            //  if revrs is positive, then accel toward something
                                            //  if revrs is negative, then accel away from something
                                            //  as revrs gets bigger, accel faster
    int dst;                                // holds temporary distance values

    // set the old position to equal the current position

    if (YOS > 20) exit = true;
    if ((sepCount > sepThresh)&&(YOS > 8)) exit = true;
    xTail = xPos;
    yTail = yPos;
    revrs = -1;

    // use the real algorithm
        // cycle through this agent's neighbors to determine how far each neighbor is away from
        //  the current neighbor and use this information to determine in what direction to travel
        // specifically cycle through each neighbor and then update the velocity of the agent
        //  based on each update (the updates are cumulative)
        //    this means that there is a possibility of numerous occurances like:
        //      the two velocity changes could be in the same direction
        //      the two velocity changes could be in opposite directions
        //      the two velocity changes could be some combination of the above two

        for (int i=0;i<numNbors;i++)
        {
            dst = distance(neighbors[i]);       // = the distance to the ith neighbor
            if (dst == 0)
            {
                // then the current neighbor is on top of this agent, so don't adjust this agent's
                //  speed and just let them "pass through" each other
                //    if revrs were changed to -1 then the agents would congregate in the lower-right corner
                //    if revrs were changed to +1 then the agents would congregate in the upper-left corner
                revrs = 0;
            }
            else if (dst < revrsdist)
            {
                // then the current neighbor is in this agent's "personal space"
                //  therefore the current agent needs to back off by reversing directions
                revrs = -1;
            }
            else
            {
                // then this agent is too far away from its neighbor
                //  so make it head back towards its neighbor
                revrs = 1;
            }
            // now adjust the velocity in the x and y directions based on the revrs parameter
            if (xPos<neighbors[i].xPos)
            {
                // then move in the positive x direction because the agent's neighbor is in front of it
                xSpeed = xSpeed + accel*revrs;
            }
            else
            {
                // move in the negative x direction because the agent's neighbor is behind it
                xSpeed = xSpeed - accel*revrs;
            }
            if (yPos<neighbors[i].yPos)
            {
                // then move in the positive y direction because the agent's neighbor is in front of it
                ySpeed = ySpeed + accel*revrs;
            }
            else
            {
                // move in the negative y direction because the agent's neighbor is behind it
                ySpeed = ySpeed - accel*revrs;
            }
        } // end for loop

        // make sure none of the parameters exceeded their maximum allowable values

        // check the speeds
        if (xSpeed > maxSpeed) xSpeed = maxSpeed;
        if (xSpeed < -maxSpeed) xSpeed = -maxSpeed;
        if (ySpeed > maxSpeed) ySpeed = maxSpeed;
        if (ySpeed < -maxSpeed) ySpeed = -maxSpeed;

        // ensure the agents stay on the canvas
        if (xPos < 10) xSpeed = bounceSpeed;
            // the agent needs to move in the positive x direction
        if (xPos > canvasWt-10) xSpeed = -bounceSpeed;
            // the agent needs to move in the negative x direction
        if (yPos < 10) ySpeed = bounceSpeed;
            // the agent needs to move in the positive y direction
        if (yPos > canvasHt-10) ySpeed = -bounceSpeed;
            // the agent needs to move in the negative y direction

        if(fitness<0.2)                         // then the agents are ready to get out
        {
            // pull the agents towards the lower-right hand corner of the screen
            //  this represents the separation zone
            if (xPos > canvasWt/4) xSpeed = xSpeed - accelToSep;
                // the agent needs to accelerate to the lower-right hand corner in the x direction
            if (yPos < 3*canvasHt/4) ySpeed = ySpeed + accelToSep;
                // the agent needs to accelerate to the lower-right hand corner in the y direction
        }

        if((fitness>0.2)&&(fitness<0.8))        // then the agents are undecided and should stay in the middle
        {
            // keep the agents closer to the center by adjusting their speeds if they get too far out
            if (xPos < canvasWt/2) xSpeed = xSpeed + accelToMid;
                // the agent needs to accelerate to the middle in the x direction
            if (xPos > canvasWt/2) xSpeed = xSpeed - accelToMid;
                // the agent needs to accelerate to the middle in the x direction
            if (yPos < canvasHt/2) ySpeed = ySpeed + accelToMid;
                // the agent needs to accelerate to the middle in the x direction
            if (yPos > canvasHt/2) ySpeed = ySpeed - accelToMid;
                // the agent needs to accelerate to the middle in the x direction
        }

        if(fitness>0.8)                         // then the agents are more willing to commit
        {
            // pull the agents towards the upper-left hand corner of the screen
            //  this represents the retention zone
            if (xPos < 3*canvasWt/4) xSpeed = xSpeed + accelToRet;
                // the agent needs to accelerate to the upper-left hand corner in the x direction
            if (yPos > canvasHt/4) ySpeed = ySpeed - accelToSep;
                // the agent needs to accelerate to the upper-left hand corner in the y direction
        }

        // give the agents a kick if they are hanging out by the walls
        if (xPos < 20) xSpeed = xSpeed + bounceSpeed/4;
            // the agent needs to move in the positive x direction
        if (xPos > canvasWt-20) xSpeed = xSpeed - bounceSpeed/4;
            // the agent needs to move in the negative x direction
        if (yPos < 20) ySpeed = ySpeed + bounceSpeed/4;
            // the agent needs to move in the positive y direction
        if (yPos > canvasHt-20) ySpeed = ySpeed - bounceSpeed/4;
            // the agent needs to move in the negative y direction

        // actually move the agent by changing its current x and y position
        xPos = xTail + (int) xSpeed;
        yPos = yTail + (int) ySpeed;

        // age each agent
        YOS = YOS + .01;
        if (YOS < 2) agentColor = Color.yellow;
        if ((YOS >=2) && (YOS < 4)) agentColor = Color.orange;
        if ((YOS >=4) && (YOS < 12)) agentColor = Color.green;
        if ((YOS >=12) && (YOS < 16)) agentColor = Color.blue;
        if ((YOS >=16) && (YOS < 20)) agentColor = Color.red;

        // update the sepCount counter
        if ((xPos < canvasWt/4) && (yPos > 3*canvasHt/4)) sepCount = sepCount + 1;

            // idea - how about adding back (lowering the sepCount) for being very happy???

} // end ProcessAgent() method
```

```
    public void Draw(Graphics g)
    {
        // the fillOval(x,y,width,height) method draws an oval tangent to the sides of
        //  a rectangle that starts at x,y and has the appropriate width and height
        // therefore in order to ensure that the tail comes out of the center of the oval
        //  it was necessary to make the oval start half way from the start of the tail
        //  and thus ensure that the tail started in the center of the circle

        g.setColor(agentColor);
        g.fillOval(xPos-2, yPos-2, 4, 4);           // draw the new oval
        g.drawLine(xTail, yTail, xPos, yPos);       // draw the tail
    } // end Draw(Graphics g) method

} // end Agent class
```

B.3. SummaryStats Class

```
// Capt Marty Gaupp's Air Force Institute of Technology Thesis Project
//
// last revised on 25 Feb 1999
//
// this is the summary statistics class used to collect and display summary statistics
// this class displays the summary statistics as the program runs
//  it keeps track of the statistics by using a vector which can grow and shrink dynamically
//  the problem is, a vector can only hold Objects, not primitive types, so each member in the vector
//   is first recreated as a wrapper-class object (of type Integer)
//  this is then stored in the vector
//  when the data needs to be retrieved, it is retrieved as a wrapper-class object (of type Integer)
//   and then re-converted using the intValue() method to a primitive integer type value
//  the integers are then stored in an array which is used by drawPolyline to draw the graph

import java.awt.*;
import java.util.*;
import java.io.*;

class SummaryStats extends Canvas
{
    // class-wide variables

    double percentExitMin;                          // the percent that exited at the minimum time allowable (8 YOS)
    double percentExitMid;                          // the percent that exited at the middle of their career (14 YOS)
    double percentExitEnd;                          // the percent that exited at the end of their career (20 YOS)
    int prcExitMin;                                 // integer representation of percentExitMin
    int prcExitMid;                                 // integer representation of percentExitMid
    int prcExitEnd;                                 // integer representation of percentExitEnd
    int counter = 0;                                // counts the number of times the paint() method is called
    int xcounter = 0;                               // counts the number of X points to be drawn
    Vector YExitMin = new Vector();                 // vector of the Y points for the minimum time allowable
    Vector YExitMid = new Vector();                 // vector of the Y points for the middle of their career
    Vector YExitEnd = new Vector();                 // vector of the Y points for the end of their career
    int[] XPts = new int[400];                      // array to hold the most recent 400 X plot points
    int[] YxtMin = new int[400];                    // array to hold the most recent 400 YExitMin plot points
    int[] YxtMid = new int[400];                    // array to hold the most recent 400 YExitMid plot points
    int[] YxtEnd = new int[400];                    // array to hold the most recent 400 YExitEnd plot points
    int arrayRows = (PilotRetention.LOOPLENGTH/PilotRetention.DELAY);
                                                    // the number of rows in the data output array
    int arrayCols = (PilotRetention.ITERATIONS*3)+3;
                                                    // the number of columns in the data output array
    double[][] outputArray = new double[arrayRows][arrayCols];
                                                    // array to hold the data for file output
    int rowNumber;                                  // current row number of the data output array
    int colNumber;                                  // current column number of the data output array
    int NumIter = PilotRetention.ITERATIONS;        // number of iterations per parameter setting

    // constructor

    public SummaryStats()
    {
        resize(400,125);
        setBackground(Color.black);
        setForeground(Color.white);
        XPts[0] = 0;
        YxtMin[0] = 105;
        YxtMid[0] = 105;
        YxtEnd[0] = 105;

    if (PilotRetention.FILEOUT)                     // print number of rows and columns in array
    {
        System.out.println("number of rows in array = " + arrayRows);
        System.out.println("number of columns in array = " + arrayCols);
    }

    } // end SummaryStats constructor

    public void paint(Graphics g)
    {
        // draw the caption
        g.drawString("Percent Sep's at:",5,118);
        g.setColor(Color.red);
        g.drawString("min YOS in Red",103,118);
        g.setColor(Color.yellow);
        g.drawString("mid YOS in Yellow",198,118);
        g.setColor(Color.green);
        g.drawString("20 YOS in Green",303,118);

        counter = counter + 1;
        if (PilotRetention.totalNumberExited == 0) percentExitMin = 0.0;
        else percentExitMin = 100.0*((double) PilotRetention.totalExitMin)/((double) PilotRetention.totalNumberExited);
        if (PilotRetention.totalNumberExited == 0) percentExitMid = 0.0;
        else percentExitMid = 100.0*((double) PilotRetention.totalExitMid)/((double) PilotRetention.totalNumberExited);
        if (PilotRetention.totalNumberExited == 0) percentExitEnd = 0.0;
        else percentExitEnd = 100.0*((double) PilotRetention.totalExitEnd)/((double) PilotRetention.totalNumberExited);
```

```
if (counter%PilotRetention.DELAY == 0)
{
    xcounter = xcounter + 1;
    colNumber = PilotRetention.COL;
    if (colNumber<0)
    {
        colNumber = 0;
        System.out.println("***** Had to reset the column number! ****");
    }
    rowNumber = PilotRetention.ROW;
    if (rowNumber<0)
    {
        rowNumber = 0;
        System.out.println("***** Had to reset the row number! ****");
    }
    prcExitMin = 105 - (int) percentExitMin;
    Integer percentExtMin = new Integer(prcExitMin);    // need to use wrapper-classes for vectors
    prcExitMid = 105 - (int) percentExitMid;
    Integer percentExtMid = new Integer(prcExitMid);    // need to use wrapper-classes for vectors
    prcExitEnd = 105 - (int) percentExitEnd;
    Integer percentExtEnd = new Integer(prcExitEnd);    // need to use wrapper-classes for vectors

    if (xcounter < 400)                      // just add an element to the end of the vector
    {
        // add the element to the end of each vector
        YExitMin.addElement(percentExtMin);
        YExitMid.addElement(percentExtMid);
        YExitEnd.addElement(percentExtEnd);
        // copy the data into the corresponding arrays
        XPts[xcounter] = xcounter;
        Integer curYExitMin = (Integer) YExitMin.elementAt(xcounter-1);
        YxtMin[xcounter] = curYExitMin.intValue();
        Integer curYExitMid = (Integer) YExitMid.elementAt(xcounter-1);
        YxtMid[xcounter] = curYExitMid.intValue();
        Integer curYExitEnd = (Integer) YExitEnd.elementAt(xcounter-1);
        YxtEnd[xcounter] = curYExitEnd.intValue();
    } // end if for the first 400 observations
    else                                     // shuffle the vector one position
    {
        // remove the first element of each vector
        YExitMin.removeElementAt(0);
        YExitMid.removeElementAt(0);
        YExitEnd.removeElementAt(0);
        // add the new element to the end of each vector
        YExitMin.addElement(percentExtMin);
        YExitMid.addElement(percentExtMid);
        YExitEnd.addElement(percentExtEnd);
        // copy the data into their corresponding arrays using a loop
        for(int i=0;i<399;i++)
        {
            Integer curYExitMin = (Integer) YExitMin.elementAt(i);
            YxtMin[i] = curYExitMin.intValue();
            Integer curYExitMid = (Integer) YExitMid.elementAt(i);
            YxtMid[i] = curYExitMid.intValue();
            Integer curYExitEnd = (Integer) YExitEnd.elementAt(i);
            YxtEnd[i] = curYExitEnd.intValue();
        }
    } // end else to handle vector shuffling
    if (PilotRetention.FILEOUT)              // output the data to a file
    {
        // put the original data into the array
        outputArray[rowNumber][3*colNumber] = percentExitMin;
        outputArray[rowNumber][3*colNumber+1] = percentExitMid;
        outputArray[rowNumber][3*colNumber+2] = percentExitEnd;
        // calculate the mean if the current column is the last column of data to be collected
        if(colNumber == (NumIter-1))
        {
            for (int column=0;column<3;column++)
            {
                double rowTotal = 0.0;
                for (int obsNum = 0;obsNum<NumIter;obsNum++)
                {
                    int colPos = column + (obsNum*3);
                    rowTotal = rowTotal + outputArray[rowNumber][colPos];
                }
                outputArray[rowNumber][3*colNumber+3+column] = rowTotal/NumIter;
            }
        } // end calculation of the mean
        // print out the data if the current column is the last column of data to be collected
        if(colNumber == (NumIter-1))
        {
            PrintStream statsPS = new PrintStream(PilotRetention.dataOut);
            String dataLine = new String(outputArray[rowNumber][0] + "\t" +
                                         outputArray[rowNumber][1] + "\t" +
                                         outputArray[rowNumber][2] + "\t" +
                                         outputArray[rowNumber][3] + "\t" +
                                         outputArray[rowNumber][4] + "\t" +
                                         outputArray[rowNumber][5] + "\t" +
                                         outputArray[rowNumber][6] + "\t" +
                                         outputArray[rowNumber][7] + "\t" +
                                         outputArray[rowNumber][8] + "\t" +
                                         outputArray[rowNumber][9] + "\t" +
                                                        outputArray[rowNumber][10] + "\t" +
                                                        outputArray[rowNumber][11] + "\t" +
                                                        outputArray[rowNumber][12] + "\t" +
                                                        outputArray[rowNumber][13] + "\t" +
                                                        outputArray[rowNumber][14] + "\t" +
                                                        outputArray[rowNumber][15] + "\t" +
                                                        outputArray[rowNumber][16] + "\t" +
                                                        outputArray[rowNumber][17]);
                    statsPS.println(dataLine);
                } // end printing out of data
            } // end if for file output
            PilotRetention.ROW = PilotRetention.ROW + 1;
        } // end if for time delay

        if (xcounter < 400)
        {
            g.setColor(Color.red);
            g.drawPolyline(XPts, YxtMin, xcounter);
            g.setColor(Color.yellow);
            g.drawPolyline(XPts, YxtMid, xcounter);
            g.setColor(Color.green);
            g.drawPolyline(XPts, YxtEnd, xcounter);
        }
        else
        {
            g.setColor(Color.red);
            g.drawPolyline(XPts, YxtMin, 399);
            g.setColor(Color.yellow);
            g.drawPolyline(XPts, YxtMid, 399);
            g.setColor(Color.green);
            g.drawPolyline(XPts, YxtEnd, 399);
        }

    } // end paint(Graphics g) method

} // end class SummaryStatistics
```

B.4. EnvironmentalControl Class

```
// Capt Marty Gaupp's Air Force Institute of Technology Thesis Project
//
// last revised on 25 Feb 1999
//
// class to define the environmental controls

import java.awt.*;
import java.awt.event.*;

public class EnvironmentControl extends Panel
{
    // class wide variables

    public ScrollBarReader hiring;               // airline hiring practices control
    public ScrollBarReader paygap;               // perceived pay-gap control
    public ScrollBarReader opstempo;             // ops tempo control
    public ScrollBarReader tmwt;                 // time vs money weighting factor control
    public DummyCanvas spacer;                   // spacer for the sliders
    public int hiringValue;                      // hiring's value
    public int paygapValue;                      // pay gap's value
    public int opstempoValue;                    // opstempo's value
    public int tmwtValue;                        // tmwt's Value

    // set up the initial values from the main program
    public int inithiring = PilotRetention.INITHIRINGVALUE;
    public int initpaygap = PilotRetention.INITPAYGAPVALUE;
    public int initopstempo = PilotRetention.INITOPSTEMPOVALUE;

    // constructor

    public EnvironmentControl()
    {
        setLayout(new GridLayout(0,1,0,-10));
        ScrollBarReader hiring = new ScrollBarReader("    Airline Job Avail", "     low", "high", inithiring, 10, 0, 100, 1, 5);
        ScrollBarReader paygap = new ScrollBarReader("    Perceived Pay Gap", "     low", "high", initpaygap, 10, 0, 100, 1, 5);
        ScrollBarReader opstempo = new ScrollBarReader("    Operations Tempo", "     low", "high", initopstempo, 10, 0, 100, 1, 5);
        ScrollBarReader tmwt = new ScrollBarReader("    Time vs Money", 100, "    tm", "$", 50, 10, 0, 100, 1, 5);
        spacer = new DummyCanvas();
        add(hiring);
        add(paygap);
        add(opstempo);
        add(tmwt);
        add(spacer);
        spacer.repaint();
        hiringValue = hiring.scrollValue;
        paygapValue = paygap.scrollValue;
        opstempoValue = opstempo.scrollValue;
        tmwtValue = tmwt.scrollValue;
    } // end EnvironmentControl constructor

    public class DummyCanvas extends Canvas
    {

        // class-wide variables

        int vertSize = 1;                        // vertical size of the box

        // constructor

        public DummyCanvas()
        {
            resize(140,vertSize);
            setBackground(Color.white);
            setForeground(Color.white);
        } // end DummyCanvas constructor

    } // end class DummyCanvas

    public int getHiringValue()
    {
        hiringValue = hiring.hiringScrollValue;
        return 100-hiringValue;
    }

    public int getPaygapValue()
    {
        paygapValue = paygap.paygapScrollValue;
        return 100-paygapValue;
    }

    public int getOpstempoValue()
    {
        opstempoValue = opstempo.opstempoScrollValue;
        return 100-opstempoValue;
    }
    public int getTmwtValue()
    {
        tmwtValue = tmwt.tmwtScrollValue;
        return tmwtValue;
    }

} // end class EnvironmentControl
```

B.5. PilotControl Class

```
// Capt Marty Gaupp's Air Force Institute of Technology Thesis Project
//
// last revised on 25 Feb 1999
//
// class to define pilot controls

import java.awt.*;
import java.awt.event.*;

public class PilotControl extends Panel
{
    // class wide variables

    public UtilityCurve money;                  // money's utility curve
    public UtilityCurve timeoff;                // time off's utility curve
    public DummyCanvas spacer;                  // spacer for the sliders
    public int moneyRhoValue;                   // money's rho value
    public int timeRhoValue;                    // time's rho value
    int horSize = 140;                          // controls horizontal size of everything

    // set up the initial values from the main program
    public int initmoney = PilotRetention.INITMONEYRHO;
    public int inittime = PilotRetention.INITTIMERHO;

    // constructor

    public PilotControl()
    {
        setLayout(new GridLayout(0,1,0,10));
        UtilityCurve money = new UtilityCurve("money","money",0,100,0,1,100,20,1,10,1,horSize,initmoney);
        UtilityCurve timeoff = new UtilityCurve("timeoff","time-off",0,100,0,1,100,20,1,10,1,horSize,inittime);
        spacer = new DummyCanvas();
        add(money);
        add(timeoff);
        add(spacer);
        spacer.repaint();
        moneyRhoValue = money.rho;
        timeRhoValue = timeoff.rho;
    } // end PilotControl constructor

    public class DummyCanvas extends Canvas
    {

        // class-wide variables

        int vertSize = 1;                       // vertical size of the box

        // constructor

        public DummyCanvas()
        {
            resize(horSize,vertSize);
            setBackground(Color.white);
            setForeground(Color.white);
        } // end DummyCanvas constructor

    } // end class DummyCanvas

    public int getMoneyRhoValue()
    {
        moneyRhoValue = money.moneyRho;
        return moneyRhoValue;
    }

    public int getTimeRhoValue()
    {
        timeRhoValue = timeoff.timeRho;
        return timeRhoValue;
    }

} // end class PilotControl
```

B.6. UtilityCurve Class

```
// Capt Marty Gaupp's Air Force Instituite of Technology Thesis Project
//
// last revised on 25 Feb 1999
//
// this class will create a utility curve with rho controlled by a slider
//  the utility curve's values will range from minY to maxY
//  the utility curve's x values will range from minX to maxX
//  the utility curve will be modeled with an exponential curve
//  rho will be allowed to range from -rhoSize to +rhoSize
//  for this to really work rhoSize needs to be as big as max(minX,maxX)
//  the monotonicity of the utility function is controlled by monoton
//  the size of the utility curve (in pixels) is controlled by size

import java.awt.*;
import java.awt.event.*;

public class UtilityCurve extends Panel implements AdjustmentListener
{
    // UtilityCurve class wide variables

    public Scrollbar rhoSlider;                   // slider for rho
    public int initialValue;                      // slider's initial value
    public int sliderSize;                        // slider's size
    public int minValue;                          // slider's minimum value
    public int maxValue;                          // slider's maximum value
    public int unitSize;                          // slider's unit size
    public int blockSize;                         // slider's block size
    public int sliderValue;                       // current value of the slider
    public UtilityCanvas utilCurveCanvas;         // utility canvas
    public int size;                              // the size of the square utility canvas
    public String curveName;                      // utility curve's name
    public String curveTitle;                     // utility curve's title to be displayed on screen
    public int minX;                              // minimum value for X axis
    public int maxX;                              // maximum value for X axis
    public int minY;                              // minimum value for Y axis
    public int maxY;                              // maximum value for Y axis
    public int negRho;                            // biggest negative rho
    public int posRho;                            // biggest positive rho
    public int rho;                               // rho's value
    public int monoton;                           // utility curve's monotonicity
                                                  // < 0 means decreasing
                                                  // > 0 means increasing
    static public int moneyRho;                   // current value of money's rho
    static public int timeRho;                    // current value of time's rho
        // these two variables need to be static so that they can be referenced outside of this class

    // constructor

    public UtilityCurve(String name, String title, int minimumX, int maximumX, int minimumY, int maximumY,
                        int rhoSize, int slider, int unit, int block, int mono, int sz, int initValue)
    {
        // set the utility curve's values

        curveName = name;
        curveTitle = title;
        minX = minimumX;
        maxX = maximumX;
        minY = minimumY;
        maxY = maximumY;
        negRho = -rhoSize;
        posRho = rhoSize;
        sliderSize = slider;
        unitSize = unit;
        blockSize = block;
        monoton = mono;
        size = sz;
        minValue = 0;
        maxValue = (posRho - negRho) + sliderSize;
        initialValue = 0 - negRho;
        sliderValue = initValue - negRho;
        rhoSlider = new Scrollbar(Scrollbar.HORIZONTAL, initialValue, sliderSize, minValue, maxValue);
        rhoSlider.setUnitIncrement(unitSize);
        rhoSlider.setBlockIncrement(blockSize);
        rhoSlider.addAdjustmentListener(this);
        ConstructCurve();
        if (curveName == "money")
            moneyRho = rho;
        if (curveName == "timeoff")
            timeRho = rho;
    } // end UtilityCurve constructor
```

```
public void adjustmentValueChanged(AdjustmentEvent e)
{
    sliderValue = e.getValue();
    if(monoton<0)                          // set rho for monotonically decreasing utility curves
    {
        if(sliderValue == initialValue)
        {
            rho = 0;
        }
        else if(sliderValue < initialValue)
        {
            rho = -1 - sliderValue;
        }
        else if(sliderValue > initialValue)
        {
            if(sliderValue <= 2*posRho)
            {
                rho = sliderValue - 2*(sliderValue - initialValue) + 1;
            }
            else if(sliderValue > 2*posRho)
            {
                rho = 1;
            }
        }
    }
    if(monoton>0)                          // set rho for monotonically increasing utility curves
    {
        if(sliderValue == initialValue)
        {
            rho = 0;
        }
        else if(sliderValue < initialValue)
        {
            rho = sliderValue + 1 ;
        }
        else if(sliderValue > initialValue)
        {
            if(sliderValue <= 2*posRho)
            {
                rho = sliderValue - 2*(initialValue) - 1;
            }
            else if(sliderValue > 2*posRho)
            {
                rho = -1;
            }
        }
    }
    utilCurveCanvas.repaint();
    if (curveName == "money")
        moneyRho = rho;
    if (curveName == "timeoff")
        timeRho = rho;

} // end adjustmentValueChanged(AdjustmentEvent e) method

// inner class ConstructCurve for constructing the utility curve

public void ConstructCurve()
{
    rho = sliderValue + negRho;
    utilCurveCanvas = new UtilityCanvas();
    setLayout(new BorderLayout());
    add(utilCurveCanvas, BorderLayout.CENTER);
    add(rhoSlider, BorderLayout.SOUTH);
} // end ConstructCurve() method

// inner class UtilityCanvas for drawing on the utility curve

public class UtilityCanvas extends Canvas
{

    // constructor

    public UtilityCanvas()
    {
        resize(size,size);
        setBackground(Color.white);
        setForeground(Color.black);
    } // end UtilityCanvas constructor

    // set up the canvas
    public void paint(Graphics g)
    {
        // draw box outline
        g.drawRect(0,0,size-1,size-1);
        g.drawRect(1,1,size-3,size-3);
        g.drawRect(2,2,size-5,size-5);
        Font fontdef = new Font("Dialog", Font.BOLD, 15);
        g.setFont(fontdef);
        g.drawString(curveTitle,25,25);
        g.drawString("rho = " + rho,25,50);
```

```
            // set up the plot point array
            int ssize = size - 1;
            int[] Xpt = new int[ssize];
            int[] Ypt = new int[ssize];
            double[] Xintrmd = new double[ssize];
            double[] Yintrmd = new double[ssize];
            double dblrho = (double) rho;
            double dblminX = (double) minX;
            double dblmaxX = (double) maxX;
            for(int i = 0; i<ssize; i++)
            {
                if(monoton<0)                    // monotonically decreasing
                {
                    if(rho == 0)                 // straight line
                    {
                        Xpt[i] = i;
                        Ypt[i] = i;
                    }
                    if(rho != 0)                 // exponential curve
                    {
                        Xpt[i] = i;
                        Xintrmd[i] = (double)i*((double)((double)(maxX-minX))/ssize);
                        Yintrmd[i] = ssize*((1-Math.exp(-1*(dblmaxX - Xintrmd[i])/dblrho))/(1-Math.exp(-1*(dblmaxX-dblminX)/dblrho)));
                        Ypt[i] = ssize-(int)Yintrmd[i];
                    }
                }
                if(monoton>0)                    // monotonically increasing
                {
                    if(rho == 0)                 // straight line
                    {
                        Xpt[i] = i;
                        Ypt[i] = ssize-i;
                    }
                    if(rho != 0)                 // exponential curve
                    {
                        Xpt[i] = i;
                        Xintrmd[i] = (double)i*((double)((double)(maxX-minX))/ssize);
                        Yintrmd[i] = ssize*((1-Math.exp(-1*(Xintrmd[i] - dblminX)/dblrho))/(1-Math.exp(-1*(dblmaxX-dblminX)/dblrho)));
                        Ypt[i] = ssize-(int)Yintrmd[i];
                    }
                }
            }

            // draw the polyline
            g.setColor(Color.red);
            g.drawPolyline(Xpt, Ypt, Xpt.length);

        } // end paint(Graphics g) method

    } // end class UtilityCanvas

    public double getYvalue(int Xval)
    {
        int ssize = size-1;
        double dblrho = (double) rho;
        double dblminX = (double) minX;
        double dblmaxX = (double) maxX;
        double Xvalue = (double) Xval;
        double Yvalue = 0.0;
        if(monoton<0)
        {
            if(rho == 0)
            {
                Yvalue = (dblmaxX - Xvalue)/(dblmaxX - dblminX);
            }
            if(rho != 0)
            {
                Yvalue = (1-Math.exp(-1*(dblmaxX - Xvalue)/dblrho))/(1-Math.exp(-1*(dblmaxX-dblminX)/dblrho));
            }
        }
        if(monoton>0)
        {
            if(rho == 0)
            {
                Yvalue = (Xvalue - dblminX)/(dblmaxX - dblminX);
            }
            if(rho != 0)
            {
                Yvalue = (1-Math.exp(-1*(Xvalue - dblminX)/dblrho))/(1-Math.exp(-1*(dblmaxX-dblminX)/dblrho));
            }
        }
        return Yvalue;

    } // end getYvalue(int Xval) method

} // end class UtilityCurve
```

B.7. ScrollBarReader Class

```
// Capt Marty Gaupp's Air Force Institute of Technology Thesis Project
//
// last revised on 25 Feb 1999
//
// this class will display a scroll bar slider with a title and its value (both centered)
//  the slider will range from min to max-sliderSize
//  to get the slider to span the whole window, use the border layout in the class that calls this class

import java.awt.*;
import java.awt.event.*;

public class ScrollBarReader extends Panel implements AdjustmentListener
{
    // class variables

    public Scrollbar control;                      // scroll bar for adjusting values
    public Label title;                            // label for the scroll bar title
    public Label leftTitle;                        // label for the left extreme of the scroll bar
    public Label rightTitle;                       // label for the right extreme of the scroll bar
    public Label value;                            // label for the scroll bar value
    public String scrollBarName;                   // name of scroll bar
    public String leftLabel;                       // label for the left extreme of the scroll bar
    public String rightLabel;                      // label for the right extreme of the scroll bar
    public int percentBase;                        // base of percentage values (generally 100)
    public int initialValue;                       // initial value of scroll bar
    public int sliderSize;                         // slider size for scroll bar
    public int minValue;                           // minimum value of scroll bar
    public int maxValue;                           // maximum value of scroll bar
    public int unitSize;                           // unit increment of scroll bar
    public int blockSize;                          // block increment of scroll bar
    public int scrollValue;                        // current value of the scroll bar
    static public int hiringScrollValue;           // current hiring value
    static public int paygapScrollValue;           // current paygap value
    static public int opstempoScrollValue;         // current opstempo value
    static public int tmwtScrollValue;             // current tmwt value
        //  these four variables need to be static so that they can be referenced outside of this class
    public boolean labels;                         // boolean variable to determine if there are extrema labels
                                                   //  = TRUE if there are extrema labels
                                                   //  = FALSE if there are not extrema labels
    public boolean percents;                       // boolean variable to determine if percents are used
                                                   //  = TRUE if percents are used
                                                   //  = FALSE if percents are not used
    public int leftPercent;                        // percent value of the left half of the scroll bar
    public int rightPercent;                       // percent value of the right half of the scroll bar

    // constructor without extra lables

    public ScrollBarReader(String name, int initial, int slider, int min, int max, int unit, int block)
    {
        // set the scrollbar's values

        scrollBarName = name;
        initialValue = initial;
        scrollValue = initial;
        sliderSize = slider;
        minValue = min;
        maxValue = max + slider;
        unitSize = unit;
        blockSize = block;
        labels = false;
        percents = false;
        if (scrollBarName == "    Airline Job Avail")
            hiringScrollValue = scrollValue;
        if (scrollBarName == "    Perceived Pay Gap")
            paygapScrollValue = scrollValue;
        if (scrollBarName == "    Operations Tempo")
            opstempoScrollValue = scrollValue;
        if (scrollBarName == "    Time vs Money")
            tmwtScrollValue = scrollValue;

        // set up the scrollbar

        scrollValue = initialValue;
        setLayout(new BorderLayout());
        control = new Scrollbar(Scrollbar.HORIZONTAL, initialValue, sliderSize, minValue, maxValue);
        control.setUnitIncrement(unitSize);
        control.setBlockIncrement(blockSize);
        control.addAdjustmentListener(this);

        // set up the title label

        Panel titlePanel = new Panel();
        titlePanel.setLayout(new FlowLayout());
        title = new Label(scrollBarName);
        titlePanel.add(title);
    // set up the value label

    Panel valuePanel = new Panel();
    valuePanel.setLayout(new FlowLayout());
    value = new Label("    " + scrollValue);
    valuePanel.add(value);

    // put the title, scrollbar, and value on the panel

    add(titlePanel, BorderLayout.NORTH);
    add(control, BorderLayout.CENTER);
    add(valuePanel, BorderLayout.SOUTH);

} // end ScrollBarReader constructor without extra labels

// constructor with left and right lables

public ScrollBarReader(String name, String left, String right, int initial, int slider,
                       int min, int max, int unit, int block)
{
    // set the scrollbar's values

    scrollBarName = name;
    leftLabel = left;
    rightLabel = right;
    initialValue = initial;
    scrollValue = initial;
    sliderSize = slider;
    minValue = min;
    maxValue = max + slider;
    unitSize = unit;
    blockSize = block;
    labels = true;
    percents = false;
    if (scrollBarName == "    Airline Job Avail")
        hiringScrollValue = scrollValue;
    if (scrollBarName == "    Perceived Pay Gap")
        paygapScrollValue = scrollValue;
    if (scrollBarName == "    Operations Tempo")
        opstempoScrollValue = scrollValue;
    if (scrollBarName == "    Time vs Money")
        tmwtScrollValue = scrollValue;

    // set up the scrollbar

    scrollValue = initialValue;
    setLayout(new BorderLayout());
    control = new Scrollbar(Scrollbar.HORIZONTAL, initialValue, sliderSize, minValue, maxValue);
    control.setUnitIncrement(unitSize);
    control.setBlockIncrement(blockSize);
    control.addAdjustmentListener(this);

    // set up the title label

    Panel titlePanel = new Panel();
    titlePanel.setLayout(new FlowLayout());
    title = new Label(scrollBarName);
    titlePanel.add(title);

    // set up the value label

    Panel valuePanel = new Panel();
    valuePanel.setLayout(new FlowLayout());
    leftTitle = new Label(leftLabel);
    rightTitle = new Label(rightLabel);
    value = new Label("" + scrollValue);
    valuePanel.add(leftTitle);
    valuePanel.add(value);
    valuePanel.add(rightTitle);

    // put the title, scrollbar, and value on the panel

    add(titlePanel, BorderLayout.NORTH);
    add(control, BorderLayout.CENTER);
    add(valuePanel, BorderLayout.SOUTH);

} // end ScrollBarReader constructor with left and right labels

// constructor with percents
```

```
public ScrollBarReader(String name, int base, int initial, int slider, int min, int max, int unit, int block)
{

    // set the scrollbar's values

    scrollBarName = name;
    percentBase = base;
    initialValue = initial;
    scrollValue = initial;
    sliderSize = slider;
    minValue = min;
    maxValue = max + slider;
    unitSize = unit;
    blockSize = block;
    labels = false;
    percents = true;
    if (scrollBarName == "    Airline Job Avail")
        hiringScrollValue = scrollValue;
    if (scrollBarName == "    Perceived Pay Gap")
        paygapScrollValue = scrollValue;
    if (scrollBarName == "    Operations Tempo")
        opstempoScrollValue = scrollValue;
    if (scrollBarName == "    Time vs Money")
        tmwtScrollValue = scrollValue;

    // set up the scrollbar

    scrollValue = initialValue;
    rightPercent = scrollValue;
    leftPercent = maxValue - scrollValue - sliderSize;
    setLayout(new BorderLayout());
    control = new Scrollbar(Scrollbar.HORIZONTAL, initialValue, sliderSize, minValue, maxValue);
    control.setUnitIncrement(unitSize);
    control.setBlockIncrement(blockSize);
    control.addAdjustmentListener(this);

    // set up the title label

    Panel titlePanel = new Panel();
    titlePanel.setLayout(new FlowLayout());
    title = new Label(scrollBarName);
    titlePanel.add(title);

    // set up the value label

    Panel valuePanel = new Panel();
    valuePanel.setLayout(new FlowLayout());
    value = new Label("     " + leftPercent + "     " + rightPercent);
    valuePanel.add(value);

    // put the title, scrollbar, and value on the panel

    add(titlePanel, BorderLayout.NORTH);
    add(control, BorderLayout.CENTER);
    add(valuePanel, BorderLayout.SOUTH);

} // end ScrollBarReader constructor with percents

// constructor with percents and left/right labels

public ScrollBarReader(String name, int base, String left, String right, int initial,
                       int slider, int min, int max, int unit, int block)
{

    // set the scrollbar's values

    scrollBarName = name;
    percentBase = base;
    leftLabel = left;
    rightLabel = right;
    initialValue = initial;
    scrollValue = initial;
    sliderSize = slider;
    minValue = min;
    maxValue = max + slider;
    unitSize = unit;
    blockSize = block;
    labels = true;
    percents = true;
    if (scrollBarName == "    Airline Job Avail")
        hiringScrollValue = scrollValue;
    if (scrollBarName == "    Perceived Pay Gap")
        paygapScrollValue = scrollValue;
    if (scrollBarName == "    Operations Tempo")
        opstempoScrollValue = scrollValue;
    if (scrollBarName == "    Time vs Money")
        tmwtScrollValue = scrollValue;
        // set up the scrollbar

        scrollValue = initialValue;
        rightPercent = scrollValue;
        leftPercent = maxValue - scrollValue - sliderSize;
        setLayout(new BorderLayout());
        control = new Scrollbar(Scrollbar.HORIZONTAL, initialValue, sliderSize, minValue, maxValue);
        control.setUnitIncrement(unitSize);
        control.setBlockIncrement(blockSize);
        control.addAdjustmentListener(this);

        // set up the title label

        Panel titlePanel = new Panel();
        titlePanel.setLayout(new FlowLayout());
        title = new Label(scrollBarName);
        titlePanel.add(title);

        // set up the value label

        Panel valuePanel = new Panel();
        valuePanel.setLayout(new FlowLayout());
        leftTitle = new Label(leftLabel);
        rightTitle = new Label(rightLabel);
        value = new Label("" + leftPercent + "  " + rightPercent);
        valuePanel.add(leftTitle);
        valuePanel.add(value);
        valuePanel.add(rightTitle);

        // put the title, scrollbar, and value on the panel

        add(titlePanel, BorderLayout.NORTH);
        add(control, BorderLayout.CENTER);
        add(valuePanel, BorderLayout.SOUTH);

    } // end ScrollBarReader constructor with percents and left/right labels

    // define the adjustment listener

    public void adjustmentValueChanged(AdjustmentEvent e)
    {
        scrollValue = e.getValue();
        if (scrollBarName == "    Airline Job Avail")
            hiringScrollValue = scrollValue;
        if (scrollBarName == "    Perceived Pay Gap")
            paygapScrollValue = scrollValue;
        if (scrollBarName == "    Operations Tempo")
            opstempoScrollValue = scrollValue;
        if (scrollBarName == "    Time vs Money")
            tmwtScrollValue = scrollValue;
        if (!labels && !percents)
            value.setText("    " + scrollValue);
        if (labels && !percents) value.setText("" + scrollValue);
        if (percents && !labels)
        {
            rightPercent = scrollValue;
            leftPercent = maxValue - scrollValue - sliderSize;
            value.setText("    " + leftPercent + "    " + rightPercent);
        }
        if (percents && labels)
        {
            rightPercent = scrollValue;
            leftPercent = maxValue - scrollValue - sliderSize;
            value.setText("" + leftPercent + "  " + rightPercent);
        }
    } // end adjustmentValueChanged(AdjustmentEvent e) method

} // end class ScrollBarReader
```

B.8. HTML Web Page Code

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<!--last modified on Tuesday, November 03, 1998 03:35 PM -->
<HTML>

<HEAD>
          <META HTTP-EQUIV="Content-Type" CONTENT="text/html;CHARSET=iso-8859-1">
          <META NAME="GENERATOR" Content="Visual Page 1.1a for Windows">
          <META NAME="Author" Content="Marty Gaupp">
          <TITLE>Capt Marty Gaupp's Thesis Project </TITLE>
</HEAD>

<BODY>

<H1 ALIGN="CENTER">Capt Marty Gaupp's Thesis Project</H1>
<CENTER>
<P><APPLET CODE="PilotRetention.class" WIDTH="700" HEIGHT="515" ALIGN="BOTTOM">
</APPLET>
</P>

<P><A HREF="mailto:mgaupp@afit.af.mil">e-mail Capt Gaupp</A>
</CENTER>

</BODY>

</HTML>
```

APPENDIX C – ANNOTATED BIBLIOGRAPHY

Bankes, Steven C. "On the Edge: Statistics and Computing". <u>Chance</u>. Winter 1994. pp. 50-51, 57.

In this article, Banks explains that the process of exploratory modeling involves comparing the results of alternative simulation models to available data. He shows that the goal of exploration is not to build a model that is the best fit for the data, but rather to better understand how the available data constrains the plausibility of alternative models. After having specified an ensemble of possible modeling experiments, any specific modeling experiment can be then thought of as a sample from this ensemble. The strategy for sampling from an ensemble of computational experiments must be driven by the question being asked. What is generally called for is not the best estimate model, but the model whose results give the best leverage in reaching a decision. The idea is to establish a threshold goodness of fit such that any model performing that well or better would be considered "interesting". It then becomes reasonable to ask for the distribution of models that perform within that threshold.

Bankes, Steven C. "Exploratory Modeling for Policy Analysis". <u>Operations Research</u>. May-June 1993. pp. 435-449.

In this article Banks defines and describes consolidative and exploratory modeling approaches. Consolidative modeling builds a model by consolidating known facts into a single predictive package. Exploratory modeling uses a series of experiments to explore the implications of varying assumptions and hypotheses regarding a system. Banks then describes how exploratory modeling can be used to gain insight into systems where little information is available for consolidative model building. The process of conducting an exploratory modeling approach is insightful in itself, the goal being to create an ensemble of models from which to pull specific cases that answer "what-if" questions.

Bankes, Steven C. and James Gillogly. "Exploratory Modeling – Search Through Spaces of Computational Experiments". <u>Proceedings of the Third Annual Conference on Evolutionary Programming</u>. eds. Anthony V. Sebald and Lawrence J. Fogel. River Edge, New Jersey: World Scientific Publishing Co., 1994. pp. 353-360.

This article shows that when analyzing complex and uncertain problems, computer models are used not to predict outcomes, but rather to explore spaces of possible outcomes, search for models with desired characteristics, or to support human reasoning via computation of the implications of alternative assumptions. Banks and Gillogly describe how exploratory modeling can be understood as searching or sampling over an ensemble of models that are plausible given a priori knowledge.

They claim that exploratory modeling strategy can be represented as an ensemble of possible computational experiments and a strategy for selecting individual experiments from that ensemble.

Bankes, Steven C. and James Gillogly. "Validation of Exploratory Modeling". Proceedings of the Conference on High Performance Computing 1994. eds. Adrian M. Tentner and Rick L. Stevens. San Diego, CA: The Society for Computer Simulation, 1994. pp. 382-387.

In this article Banks and Gillogly show how exploratory modeling is primarily useful for situations where insufficient information exists to build a truthful model of the system of interest. They point out that the problem of how to cleverly select the finite sample of models and cases to examine from the infinite set of possibilities is the central problem of exploratory modeling methodology. Thus, in exploratory modeling, rather than validate models, one must validate research strategies. This validation centers on three aspects of the analysis:

1. the specification of an ensemble of models that will be the basis for exploration;
2. the strategy for sampling from this ensemble;
3. the logic used to connect experimental results to study conclusions.

Beyerchen, Alan. "Clausewitz, Nonlinearity, and the Unpredictability of War". International Security. Winter 1992/1993. pp. 59-90.

In this article, Beyerchen explains how Clausewitz's On War is suffused with the understanding that every war is inherently a nonlinear phenomenon, the conduct of which changes its character in ways that cannot be analytically predicted. He shows that due to these nonlinearities, our ability to predict the course and outcome of any given conflict is severely limited. Clausewitz points out that unpredictability from chance comes in three guises:

1. statistically random phenomenon;
2. the amplification of a microcause;
3. a function of our analytical blindness.

Beyerchen shows that Clausewitz knew about nonlinearity and chaos, but that he just didn't have the vocabulary available to explain it.

Boehm, Barry W. "A Spiral Model of Software Development and Enhancement". Computer. May 1988. pp 61 – 72.

In this article, Boehm describes the spiral model of software development, which includes the following 4 cyclic steps:

1. determine objectives, alternatives, and constraints;
2. evaluate alternatives, identify risks, and resolve risks;
3. develop and verify next-level product;
4. plan the next phase.

He shows that each of the above steps is carried out in a cyclic fashion until the finished product is arrived at (or never if the finished product is to be continuously updated to fit the user's ever changing needs).

Caryl, Matthew. "Swarm". Swarm Java Applet. World Wide Web, http://www.catachan.demon.co.uk/Alife/swarm.html. 20 August 1998.

Caryl is the author of a Java applet called "swarm". This applet mimics the swarming and flocking behavior of various creatures found in nature. He has included an excellent user interface that takes into account each agent's sphere of observation and allows the user to alter characteristics based on proximity to other agents. The controls provided also allow the user to create either insect-like swarming behavior, bird-like flocking behavior, as well as many other possibilities in between. Overall, this is an excellent applet well worth a look.

Callander, Bruce D. "The Views of the Force". Air Force Magazine. August 1998. pp 59 – 63.

This article provides statistics on the latest poll conducted by the Chief of Staff and finds that a great many Air Force members are thinking about leaving the service.

Casti, John L. Complexification – Explaining a Paradoxical World Through the Science of Surprise. New York, NY: HarperCollins, 1994.

In this book, Casti goes over the various notions of "complexifying" simple behavior into complex systems. Although the book does provide some good information, it takes many tangents and explores lots of trivial ideas. The book, however, does provide some interesting asides and insights and has a great annotated bibliography in the back under the "To Dig Deeper" section.

Deitel, H. M. and P. J. Deitel. Java: How to Program. Upper Saddle River, NJ: Prentice Hall, 1998.

This text on Java by Deitel and Deitel is an excellent introduction to the Java language. Covering JDK 1.1 (Java Development Kit), this book provides hundreds of source code examples and pages of excellent instruction. Included with the book is a CD-ROM containing all the source code presented in the book as well as eight hours of multimedia instruction. The index is also very helpful in locating specific key words and Java specific reserved words. Overall, this is an excellent introductory text for the Java language.

Dolan, Ariel. "Artificial Life on the Web, Java Alife Experiments and Artist 3D Dolls". Floys and Artificial Life Web Site. World Wide Web, http://www.aridolan.com/. 20 August 1998.

Dolan is the author of several Java applets employing Craig Reynolds ideas about boids. Calling his creations "floys", each applet increases in complexity from a simple observation oriented complex adaptive system to a dynamic user controllable genetic algorithm based evolutionary floys applet. Each applet's source code is provided, although documentation is very limited. The graphics employed are simple yet excellent and the algorithms used in the source code are quite enlightening. This applet was used extensively in the creation of the model presented in this thesis effort. The algorithms used for controlling the complex adaptive behavior of the agents in the floys was used as a template for the complex adaptive behavior of the agents in the model presented in this thesis.

Durham, Maj Susan E. <u>Chaos Theory for the Practical Military Mind</u>. Air Command and Staff College 97-03.

In this article, Durham defines chaos and describes the various characteristics of a chaotic system:

determinism;
nonlinear;
sensitivity to initial conditions;
bounded by strange attractors.

She describes chaos theory's relevance to the military mind at length and provides many examples to back up her findings. The glossary of terms at the end of the paper is excellent and makes the entire paper much easier to understand.

Gleick, James. <u>Chaos – Making a New Science</u>. New York, NY: Viking Penguin, Inc., 1987.

This book gives an excellent history of the development of chaos theory. Starting with Edward Lorenz and the Butterfly Effect, Gleick works in information about the fractal nature of chaotic systems, examines the universal applications of chaos theory, and shows how there is order in chaos and chaos in order. He ends with an examination of the uses of chaos in modern day science including the hard sciences (physics, math, biology, and chemistry) and the soft sciences (physiology, psychology, and other social sciences). Overall this is an easy, interesting read and provides an outstanding account of the history and development of the new science of chaos theory.

the dependability of plans and expectations;
the purpose and values of combat analysis.

He claims that the essential and axiomatic property of combat is not that it is stochastic but that it is suffused with uncertainty. Because combat comprises both random and deterministic phenomena, perspective determines how much weight to place on each. A good description and understanding come before prediction. Therefore, Hughes explains, if predictive power is the aim, the reduction of risk is a reasonable goal, not the elimination of uncertainty.

Ilachinski, Andrew. <u>Irreducible Semi-Autonomous Adaptive Combat (ISAAC): An Artificial-Life Approach to Land Warfare</u>. Alexandria, VA: Center for Naval Analysis, August 1997.

In this paper, Illachinski describes the generally properties of complex adaptive systems and how they apply to combat systems:
nonlinear interaction – can't be "linearlized";
nonreductionist – simple interactions result in complex behavior;
emergent behavior – global patterns emerge from local interactions;
hierarchical structure – agents follow a hierarchy of instructions from the bottom-up;
decentralized control – each agent is autonomous;
self-organization – local chaos induces global order;
nonequilibrium order – there is no "end state";
adaptation – agents adapt to the changing environment;
collective dynamics – collectively the entire chaotic system creates order.

He presents the ISAAC model and gives several examples of how it works in simulating the complex behavior of marine fighting units.

James, Maj Glenn E. <u>Chaos Theory – The Essentials for Military Applications</u>. Naval War College: Newport Paper Number Ten, October 1996.

In this paper, James describes the necessary and sufficient conditions for a system to be considered chaotic:
bounded;
nonlinear;
non-periodic;
sensitive to small disturbances;
mixing.

He expounds on the various occurrences of chaos in military systems and shows how chaos can be used to one's advantage. At the end of his text, he provides an annotated bibliography which reviews several excellent books about chaos theory.

Holland, John H. Emergence – From Chaos to Order. Reading, MA: Addison-Wesley Publishing Company, Inc., 1998.

Holland's latest work expands on his earlier book Hidden Order. A bit esoteric, he seems to go off on too many tangents to make much sense throughout the theme of the book. The beginning and very end of the book are informative, the middle (chapters 3 through 11), however, goes off on too many tangents. Holland's prior book, Hidden Order, seems to not benefit too greatly from the information presented here.

Holland, John H. Hidden Order – How Adaptation Builds Complexity. Reading, MA: Addison-Wesley Publishing Company, Inc., 1995.

This book is an introduction to the concept of complex adaptive systems. In it, Holland defines the basic properties of complex adaptive systems:
aggregation – uses categories to establish equivalence;
nonlinearity – the whole is more than the sum of the parts;
flows – results in a multiplier effect and a recycling effect;
diversity – each agent fills a niche.
He also explains the basic mechanisms of complex adaptive systems:
tags – used in aggregation and boundary formation;
internal models – mechanisms for anticipation;
building blocks – impose regularity by allowing agents to evolve and adapt.
At the end of the book, he presents ECHO, a model of a complex adaptive system based on cellular schema and the properties and mechanisms explained above. This is an excellent introduction to complex adaptive systems theory and allows the reader to gain a good foothold into this new science.

Horton, Ivor. Beginning Java. Birmingham, United Kingdom: Wrox Press, 1997.

This is an outstanding introductory and instructive text for use in learning the Java language. Horton takes great care in explaining all the peculiarities of the Java language and using this book requires only a very limited knowledge of computer programming. There are countless examples provided, and throughout the chapters, the same examples are used over and over again as a means to show how better coding schemes can improve old programs and add a degree of finesse. Overall this is a great book to read in order to learn the Java language (JDK 1.1).

Hughes, Wayne P. "Uncertainty in Combat". Military Operations Research. Summer 1994. pp. 45-57.

In this article, Hughes shows that uncertainty in combat is a very big subject, pervading nearly every aspect of combat theory, including:
the setting in which command decisions are made;
the effectiveness of decision implementation;

Kauffman, Stuart. At Home in the Universe. New York: Oxford University Press, 1995.

This book examines the science of complexity from a biology viewpoint. Evolution is discussed at length and many suggestions are given as to how life on Earth arose from the big bang. The end of the book exposes the reader to the notions of social and economic evolution and presents examples that help explain some of today's social and economic phenomena. Overall, this is a great look at the biological ramifications of complexity, however, aside from that, there is not too much new information presented here that isn't covered in a chapter of Gleick's Chaos.

Levy, Steven. Artificial Life – A Report from the Frontier Where Computers Meet Biology. New York: Vintage Books, 1992.

In this book, Levy covers the history and development of the science of artificial life. He starts with Conway's Game of Life and goes to modern robotics and computer viruses. In the midst of all this he includes lots of discussion on the definition of life and describes the philosophical dilemma of the entire artificial life phenomena. Overall, this is a great book about the development of the science of artificial life and presents the reader with dozens of concrete examples of artificial life. It also encourages the reader to formulate his own definition of life and use it to further investigate the implications of artificial life.

McDonald, John W. "Exploiting Battlespace Transparency: Operating Inside an Opponent's Decision Cycle". War in the Information Age. eds. Robert L. Pfaltzgraff Jr. and Richard H. Shultz Jr. pp. 143-168.

This article describes the implications of chaos theory with regards to information obtained from the battlespace. It covers the OODA (Observe-Orient-Decide-Act) loop and how chaos theory can be used to better understand and exploit it.

Nicholls, Maj David and Maj Todor D. Tagarev. "What Does Chaos Theory Mean for Warfare". Airpower Journal. Fall 1994. pp. 48-57.

This article describes the implications of the presence of chaos in warfare in the computer simulation process. It shows how chaos theory can be used to define the minimum number of variables required to create a model. The fractal nature of chaotic systems may allow relatively small and simple war games to accurately simulate warfare. Added to this, sensitivity to initial conditions can help define a predictive confidence. If small changes produce small variations in the system, then predictions can be counted on more. However, if small changes produce large variations in the system, then predictions are to be suspect. By using the model to alter initial conditions, we can identify centers of gravity in the enemy's system. Nicholls also expands on the sources of nonlinearity in warfare including:

feedback loops;

the psychology associated with interpreting enemy actions;
the number of processes within warfare that appear inherently nonlinear;
Clausewitzian friction;
the process of decision making itself.

As an example of chaos theory at work, he introduces the use of fractals in image compression and map-making in the battlezone.

Reynolds, Craig. "Boids (Flocks, Herds, and Schools: a Distributed Behavioral Model)". Boids Web Site. World Wide Web, http://hmt.com/cwr/boids.html. 20 August 1998.

This is an excellent web page presenting lots of background information about Craig Reynolds and his "boids" project. The information presented includes a working applet of Reynold's boids and allows the user to experience first-hand the wonders of bird-like flocking behavior on the computer.

Simon, Herbert A. "Prediction and Prescription in Systems Modeling". <u>Operations Research</u>. January-February 1990. pp. 7-14.

This article describes the difference between prediction and prescription. With regards to prediction, a great deal of the power of engineering analyses, and their computational feasibility, resides in the fact that we do not have to track dynamic paths in detail, but only discover the stable and unstable equilibria of the system. If we change the questions that we seek to answer in our modeling, then it is also probable that we will want to change the models and our methods of analyzing them. With regards to prescriptive models, we seek to predict events we cannot control in order to adapt to them better. Simon shows that intelligent approximation, not brute force computation, is still the key to effective modeling and that prescription is planning for the future. Our practical concern in planning for the future is what we must do now to bring that future about. He claims that we must use our future goals to detect what may be irreversible present actions that must be avoided, and to disclose gaps in our knowledge that must be closed soon so that choices may be made later. Our decisions today require us to know our goals, but not the exact path along which we will reach them.

Vanderwal, Rich. "It's Alive – Boids of a Feather". Boids Web Site. World Wide Web, http://www.discovery.com/area/science/life/life1.3.html. 20 August 1998.

This web site is a great introduction to Craig Reynold's flocking creation called "boids". Lots of information is given about the theory behind the boids' behavior as well as links to many other sites about Craig Reynolds.

Vulliamy, Alex. "Alex Vulliamy's HomePage". Flies Web Site. World Wide Web, http://www.vulliamy.demon.co.uk/alex.html. 20 August 1998.

Vulliamy's applet is a great introduction to the algorithms involved in simulating insect-like swarming behavior. Short and well documented, the source code is readily available and quite easy to understand. This site is an excellent place to go for information about how to code complex adaptive systems behavior in Java.

Waldrop, M Mitchell. Complexity, the Emerging Science at the Edge of Order and Chaos. New York, NY: Touchstone, 1992.

This book covers the history of the science of complexity as it was formed at the Santa Fe Institute. Waldrop provides lots of great quotes about the science of complexity and includes tons of first-hand accounts by the founders of complex adaptive systems theory themselves. Painstakingly researched, Waldrop presents information regarding evolution, artificial life, and the economic applications of complex adaptive systems theory. Overall, this book is an excellent introduction to the science of complexity and its development over the past few decades.

Wright, Chris. Teach Yourself Java. London: Hodder Headline Plc, 1998.

This text is a great introduction to the Java language. Aimed at supplementing an already pre-existing Java knowledge base, this book helps by providing simple source code examples of various Java tasks. Among the topics covered in depth are graphics and animation, with particular attention to how Java handles the presentation of various forms of graphics. Overall, this book is a great supplement to any other more in depth Java resource, however, by itself it is a bit limited in scope.

BIBLIOGRAPHY

Beyerchen, Alan. "Clausewitz, Nonlinearity, and the Unpredictability of War". International Security. Winter 1992/1993. pp. 59-90.

Casti, John L. Complexification – Explaining a Paradoxical World Through the Science of Surprise. New York NY: HarperCollins, 1994.

Clemen, Robert T. Making Hard Decisions: An Introduction to Decision Analysis, 2nd Edition. Belmont California: Duxbury Press at Wadsworth Publishing Company, 1996.

Dolan, Ariel. "Artificial Life on the Web, Java Alife Experiments and Artist 3D Dolls". Floys and Artificial Life Web Site. World Wide Web, http://www.aridolan.com/. 20 August 1998.

Durham, Maj Susan E. Chaos Theory for the Practical Military Mind. Air Command and Staff College 97-03. Maxwell AFB Montgomery AL.

Gleick, James. Chaos – Making a New Science. New York NY: Viking Penguin, Inc, 1987.

Holland, John H. Emergence – From Chaos to Order. Reading MA: Addison-Wesley Publishing Company, Inc., 1998.

Holland, John H. Hidden Order – How Adaptation Builds Complexity. Reading MA: Addison-Wesley Publishing Company, Inc., 1995.

Hughes, Wayne P. "Uncertainty in Combat". Military Operations Research. Summer 1994. pp. 45-57.

Ilachinski, Andrew. Irreducible Semi-Autonomous Adaptive Combat (ISAAC): An Artificial-Life Approach to Land Warfare. Alexandria VA: Center for Naval Analysis, August 1997.

James, Maj Glenn E. Chaos Theory – The Essentials for Military Applications. Naval War College: Newport Paper Number Ten, October 1996.

Kauffman, Stuart. At Home in the Universe. New York: Oxford University Press, 1995.

Kirkwood, Craig W. Strategic Decision Making: Multiobjective Decision Analysis with Spreadsheets. Belmont California: Duxbury Press at Wadsworth Publishing Company, 1997.

Levy, Steven. Artificial Life – A Report from the Frontier Where Computers Meet Biology. New York: Vintage Books, 1992.

McDonald, John W. "Exploiting Battlespace Transparency: Operating Inside an Opponent's Decision Cycle". War in the Information Age: New Challenges For U.S. Security Policy. eds. Robert L. Pfaltzgraff Jr. and Richard H. Shultz Jr. International Security Studies Program, 1997. pp. 143-168.

Nicholls, Maj David and Maj Todor D. Tagarev. "What Does Chaos Theory Mean for Warfare". Airpower Journal. Fall 1994. pp. 48-57.

Reynolds, Craig. "Boids (Flocks, Herds, and Schools: a Distributed Behavioral Model)". Boids Web Site. World Wide Web, http://hmt.com/cwr/boids.html. 20 August 1998.

Waldrop, M Mitchell. Complexity, the Emerging Science at the Edge of Order and Chaos. New York NY: Touchstone, 1992.

VITA

Captain Martin P. Gaupp was born on 2 February 1972 in Summit, New Jersey. He graduated from Watchung Hills Regional High School in 1990 and entered the United States Air Force Academy in Colorado Springs, Colorado that summer. In June of 1994 he graduated with a bachelor of science degree in Economics and Operations Research. On 1 June 1994 he received his commission as an officer in the United States Air Force. In the spring of 1995 he entered the business program at Saint Martins College in Olympia, Washington, where he graduated with a Masters in Business Administration in May of 1997.

Capt Gaupp's first assignment was to McChord AFB in Tacoma, Washington. At McChord he worked in the 62d Contracting Squadron as a base contracting officer, negotiating and administrating construction and service contracts for McChord AFB. In August of 1997 he entered the School of Engineering at the Air Force Institute of Technology, Wright-Patterson AFB, Dayton, Ohio. There he pursued a Masters of Science in Operations Research.

Permanent Address: 19 Hunters Trail
Warren, NJ 07059

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1999	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE PILOT INVENTORY COMPLEX ADAPTIVE SYSTEM (PICAS): AN ARTIFICIAL LIFE APPROACH TO MANAGING PILOT RETENTION	5. FUNDING NUMBERS
6. AUTHOR(S) Martin P. Gaupp, Captain, USAF	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology 2950 P Street WPAFB OH 45433-7765	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GOR/ENS/99M-06
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Maj Craig J. Willits Air Force Personnel Operations Agency (AFPOA) 1235 Jefferson Davis Highway, Suite 301 Arlington VA 22202	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The retention of skilled pilots continues plague the U.S. Air Force. After spending millions of dollars on training and education, the mass exodus of experienced aviators from the Air Force over the past decade is disheartening. Many blame the economy, others the Air Force itself, but few are able to accurately predict how or why they are all leaving. The current personnel models do not adequately determine retention rates. Complex adaptive systems theory, however, might provide some insight. By modeling the system at the pilot's level, allowing each pilot to be represented as an autonomous, independent agent continually adapting to its environment and the other agents in it, an alternate model can be built; one that accounts for the interactions among the pilots, not just their interactions with their environment. PICAS (Pilot Inventory Complex Adaptive System) is just such a model. In PICAS, pilots 'evolve', for lack of a better word, to a greater fitness within their environment. In the process the model user can alter the environmental parameters and pilots' attitudes to determine what kind of scenarios need to be created and maintained in order to ensure that adequate retention rates are sustained.

14. SUBJECT TERMS Complex Adaptive Systems (CAS), Artificial Life, Chaos Theory, Pilot Retention, Personnel Models, Agent-Based Simulation Models			15. NUMBER OF PAGES 158 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL